Figure Extraction Website

Final Report
CS 4624 Multimedia, Hypertext, and Information Access
Virginia Tech, Blacksburg VA 24061

Professor: Edward A. Fox
Clients: Bill Ingram, Jian Wu
May 11, 2022

By:
Jonathan Reyrosa
Sa Hyun Min

Table of Contents

Table of Figures

Abstract

This project aimed to extract figures from theses and dissertations, index them, and support searching of those figures. Figure Extraction Website intends to fix the problem of users having to go through each PDF file and find figures that match their interests. Instead, Figure Extraction Website allows curators or users to upload PDF files from their computer, and then support searches by the appropriate keyword inputs. Figures can be searched by the caption text or the words within the figures. Two open source tools, PDFFigures2 and PDFPlumber, are used to extract figures from the PDF files. Then, ElasticSearch is used to index the figures, captions, and document metadata.

The website is built based on the ETDUI website, which was given to our group by our clients. ETDUI allows entry of keywords and output of PDF files that have the keyword in the title or in the summary portion of PDF files. To focus on our aims, we removed some features of ETDUI, including login, register, advanced search, and voice search. Then, our group added some features, including a file select button and an upload button, so that users can easily upload PDF files.

Currently, the website is running on localhost, which can be cloned from the GitHub repository (https://github.com/JRReynosa/CS4624_Figure_Extraction_Website). The PDF files that are uploaded are stored locally, with path information given in the website.

Testing proceeded with uploading a few PDF files. Searching was tested with keyword queries. There still exist problems, such as to find words or mathematical equations within the figures, as opposed to those within the captions.

Introduction

Objective

Our project was the result of merging two projects, and so the following were our primary objectives after the merging:

- Use open-source package(s) (PDFFigures2 and PDFPlumber) to extract figures and captions from 1000 PDF documents
- Index the figures, captions, and document metadata into Elasticsearch, and
- Build a web-based interface that
    - accepts text queries and returns the appropriate figures, figure captions and the document metadata.
    - accepts a PDF document as input and returns the figures and tables.

Several things need to happen behind the scenes for the objective to be achieved.

Deliverables

The purpose of this project is to allow for ease of access to effectively search for figures existing in an electronic thesis or dissertation (ETD). In order to achieve this, the following deliverable was produced:

1. A web-based search interface that:
    a. Accepts an ETD document (in PDF) and metadata file (in XML), extracts figures and captions from the PDF file, and indexes them into Elasticsearch
    b. Accepts text queries and returns ETD figures, figure captions, and associated ETD metadata.

Client

The clients for our project are William Ingram, an Assistant Dean and IT Services Director for Virginia Tech University Libraries, and Jian Wu, an Assistant Professor in the Computer Science Department at Old Dominion University (ODU). William Ingram is an internationally recognized digital libraries scholar specializing in computational use of library collections, specifically with an interest in ETDs. Jian Wu is the tech leader of the

CiteSeerX project as well as the lab director of ODU's Applied Machine Learning and Natural Language Processing Systems (LAMP-SYS) group. Both William Ingram and Jian Wu have published multiple papers in their respective fields, and both are quite recognized within their fields.

Team

The team for project figures is composed of Jonathan Reynosa and Sa Hyun Min.

Requirements

Open-source Extraction Routine

To extract figures, the project needed to make use of an open source package(s) to extract figures from provided ETDs in the form of PDF files. The tools suggested to be used in the project were PDFFigures2 [1] and PDFPlumber [10]. Each tool extracts a different set of figures. To resolve this, ETDs were processed through both tools, and then we accounted for duplicates within our extraction script written in Python.

Elasticsearch

Elasticsearch [4] was required to be our document store for figures, captions, and document metadata. Elasticsearch is a distributed, free and open search and analytics engine for all types of data, including textual, numerical, geospatial, structured, and unstructured. Elasticsearch was selected since it is easy to use through API calls, and for the speed and scalability of the searching services provided.

Website

The website is based on ETDUI, which was provided by our client and Winston Shields, who is an Old Dominion University graduate student. The main structure of the original website allows users to type in keywords and find PDF files that have the keyword in the title or in the summary portion of PDF files. Information on PDF files is stored as a JSON file, including identifier number, contribution and author, title, publisher, date issued, description and abstract, and URL of the PDF files.

Our group modified the website so that when users search for keywords, figures that are associated with the keywords, and occur within PDF files, are shown as a result. Furthermore, our group allowed users to select their own PDF files from their own computer and upload them to the website. Multiple PDF files can be chosen at one time, and the PDF files chosen would be indexed by Elasticsearch. The figures within PDF files users uploaded are searched when users type in keywords.

Design

Design Schema:

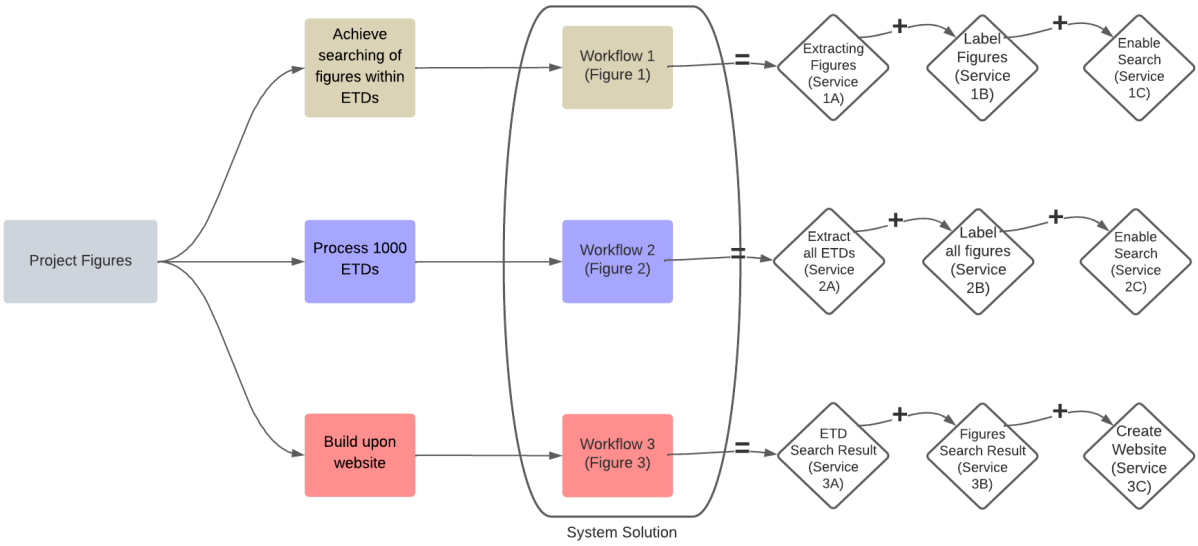

Figure 1. System Methodology Diagram

Figure 1 depicts the methodology behind our project, specifically that it was broken down into three main portions. Those are: figure extraction, ETD processing, and frontend presentation, and each is represented in Figure 1 by the tan, blue, and red workflows, respectively.

The workflow for figure extraction, depicted as Workflow 1 in Figure 1, required firstly the creation of the figure extraction script using PDFFigures2 and PDFPlumber. Figure extraction is the foundation of the project, and so Workflow 1 describes the work in the context of a single PDF. It concerns the extraction of figures for a single PDF, the indexing of the data in Elasticsearch for a single PDF and all its figures, and lastly the ability to search for the data making use of the Elasticsearch API.

As for the ETD processing portion, the requirement for our project is that as many figures as possible need to be extracted and labeled correctly and after that they should be indexed using Elasticsearch. The completion of the figure extraction workflow, depicted as Workflow 2 in Figure 1, allowed us to apply this technique to various ETDs.

Thus, we replicated this workflow for the various other ETDs provided and so we extracted and indexed hundreds of figures and their respective metadata.

Lastly, the figures and their labels should be searchable through the use of Elasticsearch's API. This requires the completion of the two aforementioned workflows as well as the updating of the frontend of the project, which is depicted as Workflow 3 in Figure 1. This workflow mentions an ETD search result as well as a Figure search result; however, the two were not able to be completed. Only the Figure Search Result was completed. The project was left at a point where only figures can be extracted and searched, but they are not linked to their respective ETD within the extracted figure metadata. The ETDUI website, described below in the ETDUI and Figure Extraction Website Design section.

All workflows and their respective responsibilities come together in our Figure Extraction Website [7] as the final product.

ETDUI and Figure Extraction Website Design:

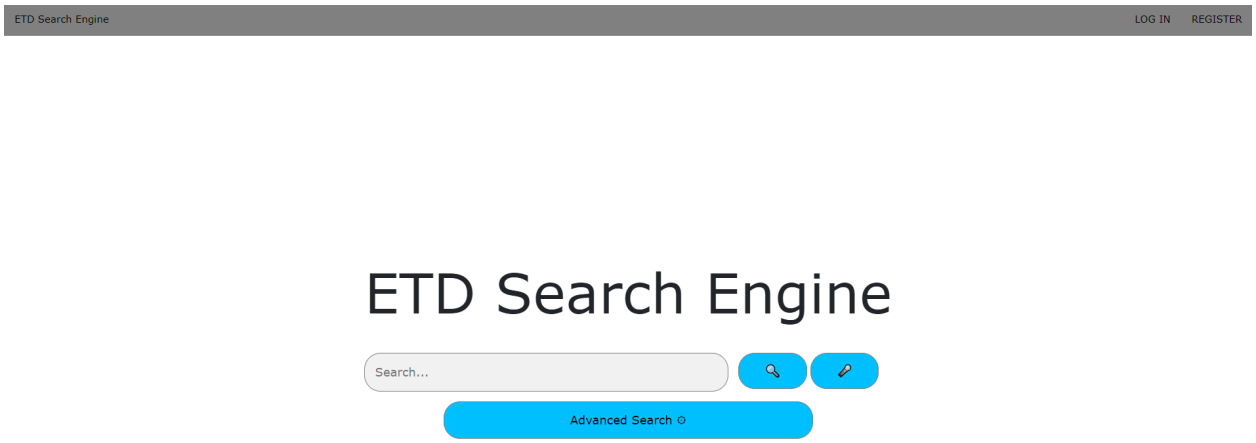

Figure 2. ETD Search Engine Main Page

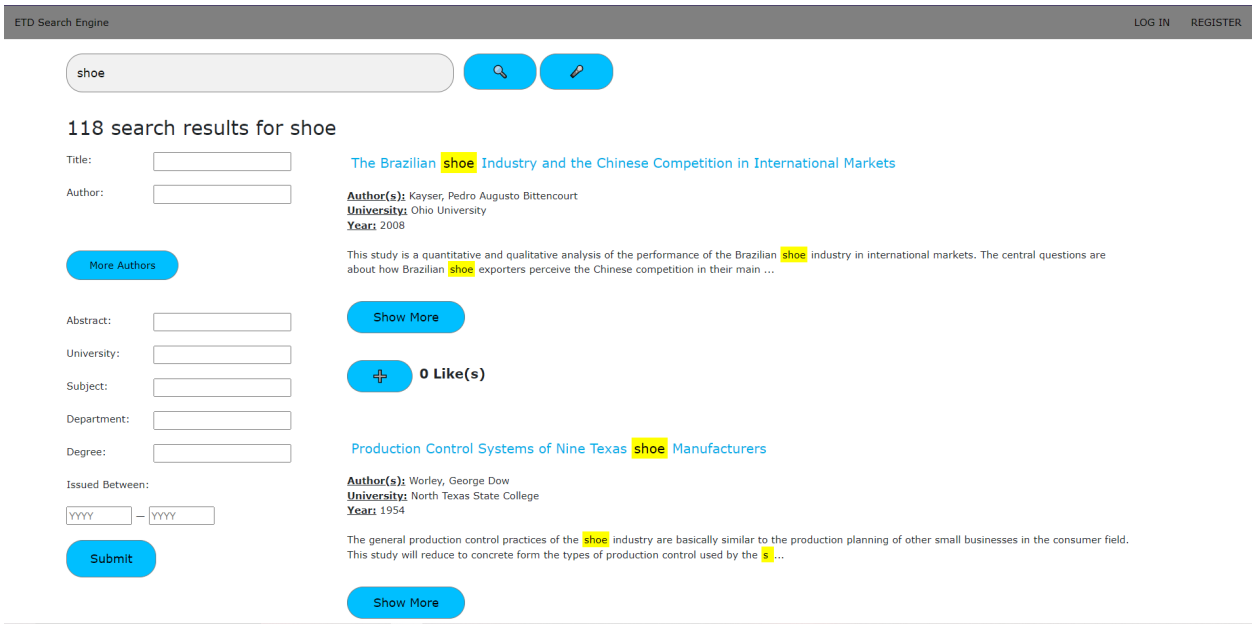

Figure 3. ETD Search Engine Result Page

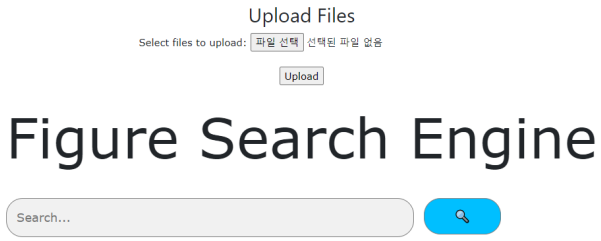

Figure 4. Figure Extraction Website Main Page

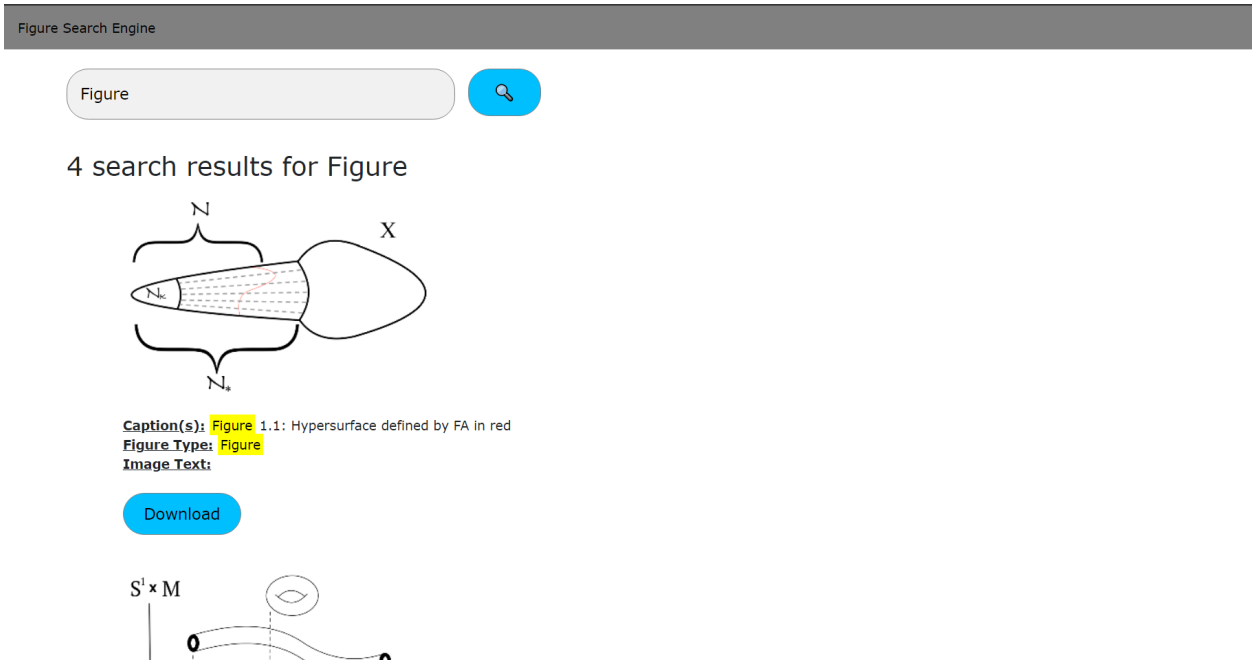

Figure 5. Figure Extraction Website Search Result Page

Figure 2 and Figure 3 show the base website of ETDUI [11], which was given to our group as reference. Figure 2 shows the main page of ETDUI, and Figure 3 shows the result page of ETDUI. When a user first accesses the ETDUI website, Figure 2 is shown. Users are then able to enter keywords for which they wish to search in the ETD database provided along with the setup of ETDUI. The website would then lead to a page similar to Figure 3, where the results are shown. In Figure 3, the user typed in “shoe” and the respective search results are shown. Users are able to see how many results are found, along with a snippet for each of the PDF files that are returned based on the user search. It also highlights the keyword in yellow so that users can easily view where the keyword was used.

In our system, the frontend is very similar to that of ETDUI and some features in ETDUI are maintained, e.g., features such as the highlighting keyword and showing the number of entries returned. However, in the case of our system the information returned regarding the entries is different, as one can see by comparing Figure 3 and Figure 5. Rather than build upon ETDUI, our group decided to create a separate project separate from ETDUI focused on the figure search. This led to the disposing of files within ETDUI, resulting in removing some functions already offered, such as advanced search

functions, login and register functions, and voice search functions. To create a system to our project needs, our group decided to add a file select button and upload button.

With all these changes, Figure 4 depicts the main page for our website. It looks very simple, and it looks similar to the original ETDUI provided to our group. Our group added a section within the main page to allow users to upload PDF files. The PDF files they upload would then be analyzed. The “choose file” button allows users to choose multiple PDF files from their computer and upload them to the website. The uploaded PDF files would be stored in the pdf_files directory within the figure_extraction directory. Figure 5 shows the result for our figure search. Our group allowed users to clearly view how many figures are returned, and the snippets of the figures that are returned. These figures are from the PDF files users uploaded from the main page of the figure extraction website, i.e., from Figure 4.

Overall, one can see the similarity between the two websites, but they ultimately serve for different purposes. ETDUI serves the purpose of searching through ETDs and their respective metadata, whereas our system searches through figures extracted from ETDs and the respective metadata of each figure. In our system, the figures and their extracted metadata do not have a connection to the ETD from which they were extracted. This issue is further discussed in the Future Work section, as creating such a connection would allow for ETD and figure search to overall create a better searching system.

Implementation

Figure Extraction, Indexing and Querying:

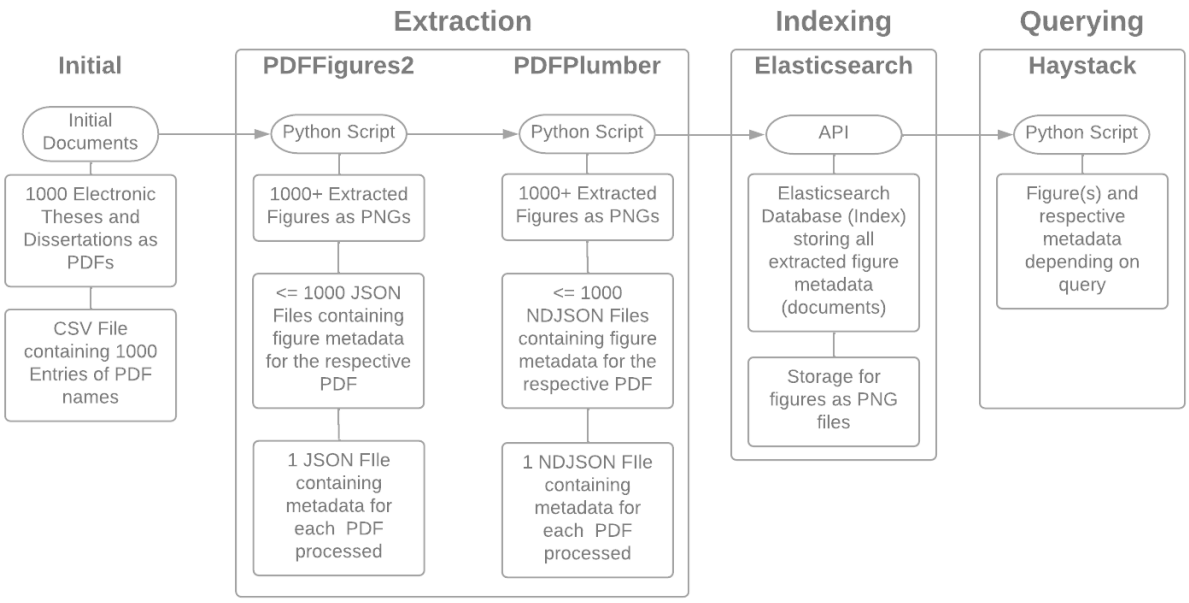

Figure 6. Pipeline for Figure Extraction, Indexing and Querying

Initially, the pipeline starts with 1000 ETDs and a CSV file containing 1000 entries with the respective name of each ETD. The ETDs are then put into our extraction pipeline that is run through a Python script and is composed of two figure extraction packages, PDFFigures2 and PDFPlumber.

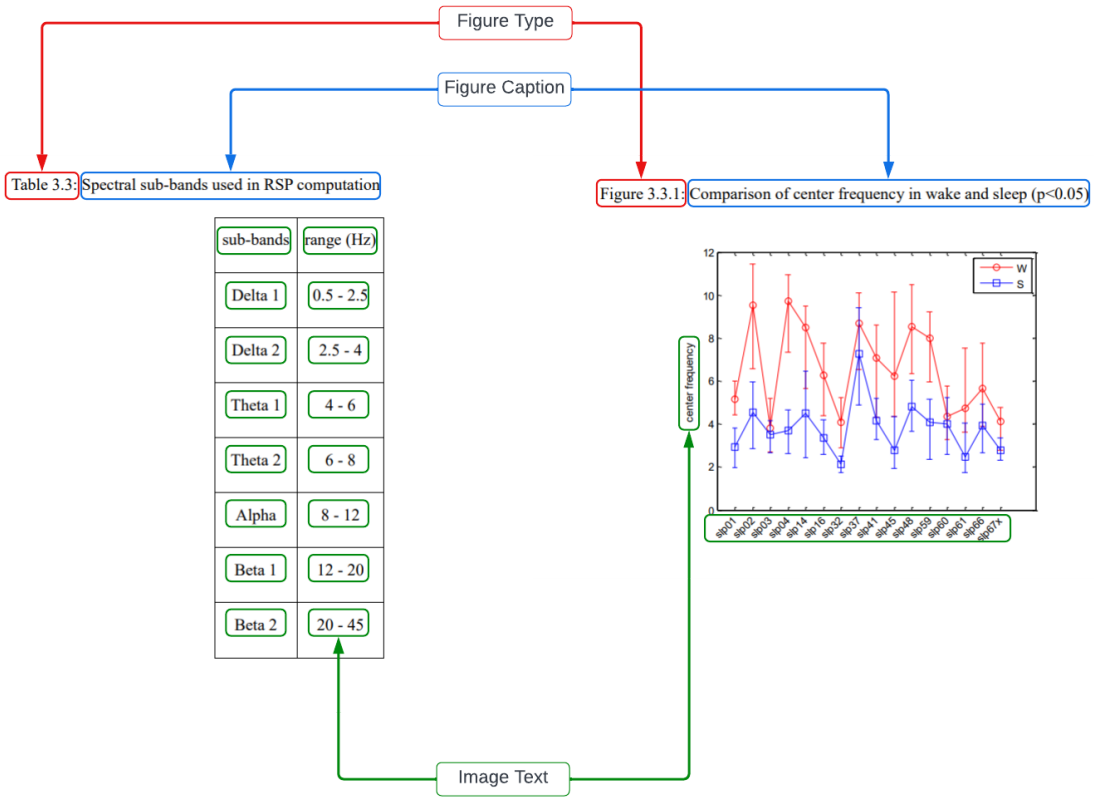

Figure 7. Extracted Figure and Metadata Example

Extraction

PDFFigures2 [1] is a command line interface tool that runs with the Scala build tool command “sbt” and takes in documents in bulk. It outputs extracted figures and metadata, such as figure caption, type, and image text, as shown in Figure 7, to a designated output folder. Thus, after PDFFigures2, there are at least 1000 extracted figures, at most 1000 JSON files containing metadata, and 1 JSON file containing metadata for each PDF processed. PDFFigures2 [1] provides usage and implementation details within their GitHub, specifically for the command in Figure 8. For further detail regarding PDFFigures2, please refer to the Developer’s Manual.

To run on lots of PDFs while saving the images, figure objects, and run statistics:

```
sbt "runMain org.allenai.pdffigures2.FigureExtractorBatchCli /path/to/pdf_directory/ -s stat_file.json -m
/figure/image/output/prefix -d /figure/data/output/prefix"
```

Figure 8. PDFFigures2 Bulk Processing Command

After the PDFFigures2 processing, we enter the second step of our extraction process, which takes in the same 1000 ETDs and once again extracts figures making use of the PDFPlumber Python library. However, the metadata from PDFFigures2 is taken into account and logic is implemented to eliminate duplicate, overlapping, and figures that are too small, making use of the bounding boxes outputted from PDFFigures2. After PDFPlumber, there is another output with at least 1000 figures, at most 1000 NDJSON files containing metadata, and 1 NDJSON file containing metadata for each PDF processed.

Indexing

Next, we enter our indexing phase where we make use of Elasticsearch. Elasticsearch uses the terms “Index” and “Documents”, which in relational database terms serve as a database and a database entry/row respectively. Elasticsearch, which serves as our document storage, uses API calls to index the metadata for each figure and its caption. Specifically, the API call we used in our script took data in bulk from output destinations of the metadata from PDFFigures2 and PDFPlumber and indexed it into our respective Elasticsearch Index named “figures”.

Querying

After all our data has been indexed in Elasticsearch, Haystack [3] was to be made of use; it was supposed to serve as a pipeline for our querying abilities. Haystack makes use of questions to query data, and so this was supposed to be reflected in our frontend. Unfortunately, we did not get to the point of implementing Haystack; this is further touched upon in the Future Work section. Since Haystack was not implemented, we used Elasticsearch API calls that compared user input from a search bar to details such as figures’ caption, type, and image text, as shown in Figure 7.

Testing and Evaluation

Figure Extraction

For the testing and evaluation of figures, the Python extraction script was run through a small subset of ETDs. This subset consisted of the first 10 ETDs out of the 1000 ETDs provided by the client for processing. The reasoning behind only 10 ETDs is that there needed to be enough data to find each portion of metadata, but not too many that it would make verifying the results difficult. For the testing of the script, the script had to take in all ETDs in a given directory, extract as many figures as possible and output them to a set output folder. As for evaluation, to evaluate a figure it had to meet the following standards:

- There exist no duplicates of this figure
- This figure is not a subset of another figure
- This figure is not too small to be understood

After all figures were extracted from testing, the figures were then manually observed to see if the logic held up these standards. This served as a continuous improvement cycle, where the logic was tweaked when needed in order to come up with our final design schema for our pipeline.

Users' manual

Developers should download everything from the Figure Extraction Website Github repository [7]. Users would have to download and install apache2, Elasticsearch, and MySQL. Run "curl -X PUT "localhost:9200/figures?pretty"" to create the Elasticsearch database. After this command, run the website by "php -S localhost:8000 -t figures". If you named the file other than "figures", change the "figures" part to whatever name you put in. Go to your browser, and you should be able to reach the website at localhost:8000. You have succeeded if you see a screen like Figure 4.

```
filename -> MATHHW8.pdf
filetempName -> /tmp/phpFT4Fgt
destFile -> /var/www/html/figures/src/figure_extraction/pdf_files/MATH HW8.pdf MATHHW8.pdf successfully uploaded

filename -> 20220419132300960.pdf
filetempName -> /tmp/phprCxGVt
destFile -> /var/www/html/figures/src/figure_extraction/pdf_files/20220419132300960.pdf 20220419132300960.pdf successfully uploaded

filename -> CkTest3.pdf
filetempName -> /tmp/phpWCo85r
destFile -> /var/www/html/figures/src/figure_extraction/pdf_files/Ck Test3.pdf CkTest3.pdf successfully uploaded

filename -> Unsafe.pdf
filetempName -> /tmp/php1FKUBs
destFile -> /var/www/html/figures/src/figure_extraction/pdf_files/Unsafe.pdf Unsafe.pdf successfully uploaded

filename -> 3342102.pdf
filetempName -> /tmp/phpzqYmxs
destFile -> /var/www/html/figures/src/figure_extraction/pdf_files/3342102.pdf 3342102.pdf successfully uploaded
```

Figure 9. File_Upload.php

Click on the "select files" button and find PDF files you want to be searched from your local computer. By clicking on PDF files while clicking the ctrl button, you are able to select multiple files. Whenever you are done choosing the PDF files, click on the "done" button. Then, click on the "Upload" button within the website. You should be able to see a page something like Figure 9. You can see from this page that all PDF files are successfully uploaded, and where the PDF files are located. The default would be in the figures/src/pdf_files directory with the PDF files named after the name of the original PDF files. The PDF files you uploaded would now be included in the results.

Developer's Manual

General

The extraction script is written in Python and makes use of the following Python packages: sys, os, json, ndjson, elasticsearch, and PDFplumber. The script itself is in the project GitHub repository [7] and its path is src/figure_extraction/extracting_script.py. It is of note that the PDFFigures2 and PDFPlumber Python packages have dependencies on other packages and programs that need to be downloaded on the system they run on.

For PDFFigures2, the machine running the command in Figure 8 needs to have the most current Java SDK. PDFFigures2 is a command line interface tool that needs to be run with the Scala build tool "sbt" command. In order to use the "sbt" command one must download it from the Scala website [12] if one is on Windows or one can use SDKMAN! in Ubuntu to install it.

For PDFPlumber, since it is a Python package, it can easily be installed using conda or pip. However, to make use of PDFPlumber's visual debugging, one must also install ImageMagick and ghostscript onto the machine. Both are software routines that are used to draw vectors on images for debugging. The most important thing to note from the software dependencies though is that they are also used to extract figures from PDF files. Without the two pieces of software, PDFPlumber can only output information of the figures but not be able to extract them.

Lastly, the Python itself is broken into a section for each extraction routine, and each section contains comments on what each line of code does. The first section runs PDFFigures2 using "sbt" and indexes the results to Elasticsearch using the Elasticsearch.helpers bulk command. The second iterates through the directory holding the ETDs and then extracts figures using the PDFPlumber Python package. For PDFPlumber, we implemented the logic and we iterated through each document in a set directory and then cross-checked each extracted figure with figures from PDFFigures2 to account for duplicates and overlapping figures.

PDFFigures2

The expected output after processing the command in Figure 8 on a single PDF is depicted in Figure 10. The -s tag in the command is for the “stat_file.json” which is the JSON file which contains the metadata for all PDF files processed. Figure 11 depicts the JSON format for the “stats_file.json” where a single PDF was processed. The -m tag is to provide the output location for extracted images which take on the naming convention of <PDF Name>-<Figure OR Table><Figure Name>-<Figure Duplicate Count>, this is depicted in Figure 11. Note the Figure Duplicate Count only updates if there are multiple figures with the same name, for instance if the PDF we processed had two figures named “Figure 1.1”, then there would be two files, where the first is “1468-Figure1.1-1” and the second is “1468-Figure1.1-2”. The -m tag also serves as the place to set the extension of the images outputted, by default PDFFigures2 outputs PNGs. The -d tag is to provide the output location for the JSON file which contains the metadata for each figure extracted from a certain PDF, and the JSON file is named after the PDF itself. The format for the JSON outputted from PDFFigures2 is depicted in Figure 12.

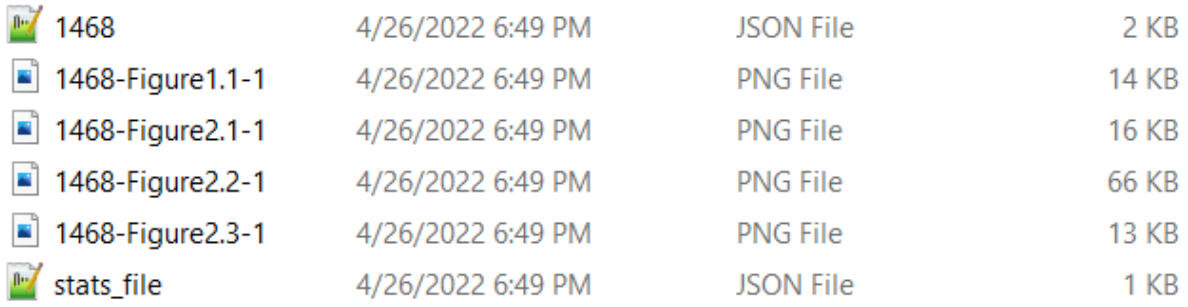

Figure 10. PDFFigures2 Expected Output


```
[{
  "filename": "C:\\Users\\jonny\\Desktop\\CS4624\\pdf\\1468.pdf",
  "numFigures": 4,
  "numPages": 100,
  "timeInMillis": 8392
}]
```


Figure 11. PDFFigures2 Stats JSON File Format

```
[{
  "caption": "Figure 2.1: Zero set of near-symplectic form in bold",
  "captionBoundary": {
    "x1": 173.0240020751953,
    "x2": 438.9722900390625,
    "y1": 257.07940673828125,
    "y2": 263.06298828125
  },
  "figType": "Figure",
  "imageText": [],
  "name": "2.1",
  "page": 50,
  "regionBoundary": {
    "x1": 205.92,
    "x2": 406.08,
    "y1": 110.88,
    "y2": 242.39999999999998
  },
  "renderDpi": 150,
  "renderURL": "C:/Users/jonny/Desktop/CS4624/pdffigures2output/1468-Figure2.1-1.png"
}, {
  "caption": "Figure 1.1: Hypersurface defined by FA in red",
  "captionBoundary": {
    "x1": 187.96099853515625,
    "x2": 424.0359191894531,
    "y1": 235.75640869140625,
    "y2": 243.53302001953125
  },
  "figType": "Figure",
  "imageText": [],
  "name": "1.1",
  "page": 26,
  "regionBoundary": {
    "x1": 219.84,
    "x2": 392.15999999999997,
    "y1": 110.88,
    "y2": 221.28
  },
  "renderDpi": 150,
  "renderURL": "C:/Users/jonny/Desktop/CS4624/pdffigures2output/1468-Figure1.1-1.png"
}, {
```


Figure 12. PDFFigures2 Data JSON File Format

Note that the PDFFigures2 extracts various metadata details from the figures, but we only made use of caption, figure type, and image text to make the figures searchable. Other details can be used for searching, to do so one can edit the code for querying which can be found in the project files stored in the GitHub repository [7] in src/elasticsearch/normal_search.php.

Elasticsearch and Kibana

Elasticsearch was installed and set up locally on the Linux machine also used for the frontend. Along with Elasticsearch, Kibana, an interface for Elasticsearch, was also

installed locally. Both services were installed locally rather than on a dedicated host or server to simplify the project as setting up an instance of both Elasticsearch and Kibana proved to be a struggle when configuring the services on a remote machine. The project was provided access to a remote Linux machine in the Virginia Tech cluster, however, network issues were met when trying to access the machine through the internet. This was possibly due to Virginia Tech restrictions to the unknown upcoming traffic from our personal computers. Due to time constraints, we did not focus further on the issue and turned to running Elasticsearch and Kibana on our personal machines. However, running Elasticsearch and Kibana on a remote server is possible and the setup is exactly the same as running it on a local machine, however, network and Elasticsearch configurations and issues should be kept in mind. Note, it is also possible to make use of Elasticsearch's service Elastic Cloud which runs Elasticsearch and Kibana in the cloud on services such as AWS, Microsoft Azure, and Google Cloud.

All indexing of documents was done through Python, but the purpose of installing Elasticsearch was to run the server locally. Ideally, an Elasticsearch server would run in the cloud. Kibana was of great use as it allowed us to debug the project from a graphical point rather than calling the Elasticsearch API to confirm the correct results. Specifically, the "Discover" and "Dev Tools" features were of great use. The former provided a graphical user interface to look through all data in our index and the latter provided an intuitive place to practice our API calls and play with our data. These two features are depicted in Figure 13 and Figure 14, respectively.

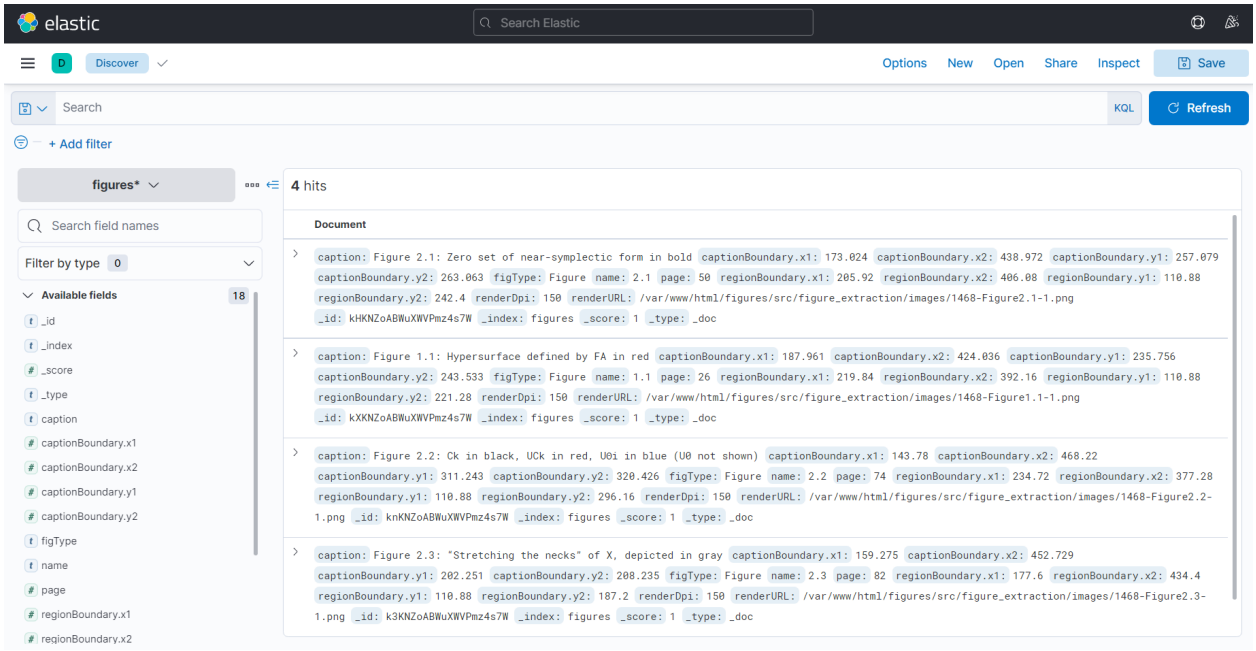

Figure 13. Kibana Index “Discover” Feature

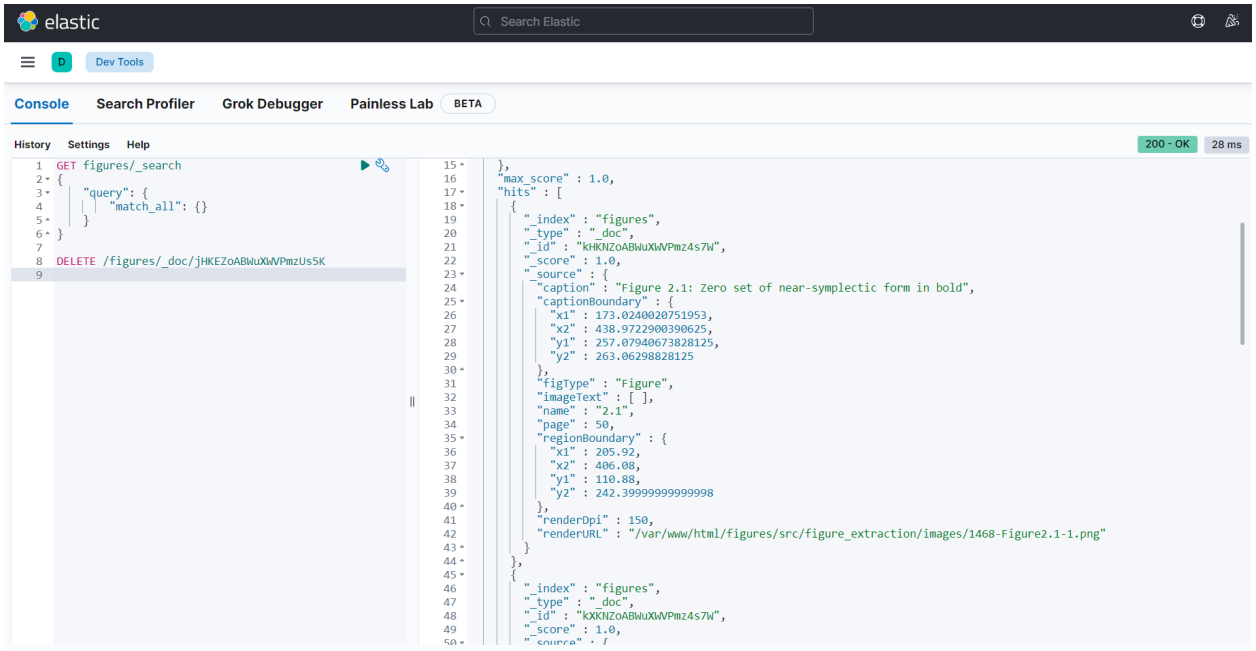

Figure 14. Kibana “Dev Tools” Feature

Lessons Learned

Timeline

Our group divided the roles into frontend and backend. Sa Hyun Min was in charge of the frontend portion of the project, and Jonathan Reynosa was in charge of the backend.

Our initial timeline is:

- 2022.02.15
  - Frontend: Get the server running. Figure out what things need to be done in the frontend.
  - Backend: Work with and write a script using PDFPlumber2 to begin extracting figures from ETDs
- 2022.03.01
  - Frontend: Get a development server. Try making very minor changes to see differences.
  - Backend: Process and extract figures from the 1000 ETDs provided
- 2022.03.15
  - Frontend: Make sure that the website can process one PDF file and its figures.
  - Backend: Index figures and respective data using Elasticsearch
- 2022.03.29
  - Frontend: Make sure that the website can process 1000 PDF files and its figures.
  - Backend: Combine frontend and backend of project in cohesive method
- 2022.04.12
  - Make changes to whatever errors are found, and improve aesthetics.
  - Backend: Clean up aesthetics and implement Haystack to better the figure searching system
- 2022.05.01
  - Frontend: Finish Project

- Backend: Finish Project

Challenges Faced

Frontend - Running Website

Running the website was harder than our group was expecting it to be. The ETDUI would only run in a Linux environment, so our group had to download Ubuntu in order to run the website. Also, with original instructions, the command to run the website was missing. Now, this part has been added with "If you ran the php -S localhost:8000 -t etdui command, access the website through http://127.0.0.1:8000/". Without this portion of the instructions, our group could not find a way to run the website even though our group has followed the instructions.

This led to emailing Winston Shields, the Old Dominion University graduate student who is in charge of the ETDUI Github repository. Getting responses, setting up a Zoom meeting, and running the website took an additional 3 weeks of time, which led to a delay of the project. Now, our group has good knowledge of how to run the website in localhost.

Backend - Pipelines

The largest issue was how to efficiently extract as many figures as possible from ETDs. Taking into mind the two packages that were suggested, PDFFigures2 and PDFPlumber, they both provide a set of results. At times, PDFFigures2 extracted figures PDFPlumber did not and vice versa. Rather than focusing on a single package it was suggested that a routine be created to merge both packages into a single pipeline making use of both. Thus, the pipeline, as explained in Figure 4, in the "Extraction" section, is to run ETDs through PDFFigures2 and then the ETD is run again through PDFPlumber. This was done in this order because PDFFigures2 runs as a Scala-based command line interface tool, and PDFPlumber is a Python package that can be used. After running through PDFFigures2, it was easier to account for duplicates, missing, etc. because those cases could be accounted for in the Python script.

Future Work

General

Future teams can work on our existing code and add on to our project. Some suggestions include hosting the website so that users can freely access the website and use the website for figure search. This would allow users to freely upload PDF files to search for figures within PDF files. Furthermore, this would help future users to skip the directions of downloading Elasticsearch and MySQL, and running the website from localhost.

Also, future teams could make a database of PDF files so that users can have an option to just search for the existing PDF files in the database. Currently, Figure Extraction Website only allows users to upload PDF files, which would then be searched. Since it is better to have more PDF files in the database, our group anticipates that having at least 10,000 PDF files would be a good starting point for the database. 10,000 PDF files would guarantee a search result for most general keywords. Furthermore, future teams can work on combining Figure Extraction Website with ETDUI, which would allow users to search for figures within the PDF files as well as words within the title or the summary portion of the PDF files. This could proceed by using the Figure Extraction Website code, and adding the word search portion of the ETDUI code into the Figure Extraction Website code.

Image Text

Most images that were indexed also had text within the image itself. Figure 8 depicts an example of figures and their respective figure details, including image text. However, we were limited to the image text that was extracted using PDFFigures2, and so image text proved to be an issue as figures could be queried with better accuracy if the image text could be accurately extracted for all images. For instance, if the table in Figure 8 did not contain the phrase “sub-bands” in its caption then it could still be queried via the “sub-bands” in the image text. Specifically, it became an issue to extract image text when the text was mathematical, contained accents, or was too blurry. This might require the introduction of another package in the “extraction” portion of our backend pipeline depicted in Figure 6 that runs optical character recognition and cross

checks that with the output of PDFFigures2 to eliminate duplicates.

Haystack

The implementation of Haystack [3] would allow for more precise searching. Haystack is a semantic, question-drived search system rather than a keyword search system which makes use of state-of-the-art natural language processing tools to allow users to search data in their natural language. For instance, in the context of the project, if one wanted to see figures depicting the U.S. economy rather than searching for “United States Economy 2022”, which makes use of keywords, one could simply type “What does the current U.S. economy look like?” into the search bar. Thus, the user could search for data, and in our case figure caption and image text, using questions, which ultimately leads to a better user experience. Haystack allows data to be stored in any database of the developer’s choice, and in our project that would be Elasticsearch, which Haystack fully supports. The frontend would not need to change in any way as the search system still functions through a search bar, but the addition of Haystack to the backend of the project would require changing the code to query the Elasticsearch data to add Haystack to pipeline which can be found in the project files stored in the GitHub repository [7] in src/elasticsearch/normal_search.php. Since the website is written using Scala and Haystack can be implemented with Elasticsearch using Python, it would perhaps be better to redesign the frontend with Python to create a more cohesive environment in the backend. Regarding Haystack itself, it was a feature that we hoped to reach, however, we failed to implement due to time constraints.

Connect Figures with their ETDs

Our implementation only extracts figures and the metadata of the extracted figure contains only data regarding the figure itself and not the ETD from which it was extracted. Our group hoped to reach a point where the Elasticsearch instance could have two clusters, or databases, one for figures and one for ETDs, and a connection could be made between them. However, the metadata provided by our extraction pipeline did not allow for such a connection as the metadata output of PDFFigures2 [1] did not contain a reference to the ETD processed. The output of PDFPlumber [10] did

provide a reference, but that reference was only the name of the ETD from which it was extracted. We chose not to use the reference to the ETD provided by PDFPlumber since only the figures extracted from PDFPlumber would have that reference. Further, since the metadata of the figures is the data stored in Elasticsearch for the respective figure, ideally the figure data entry would contain a reference to its respective ETD in the index of the ETD in the Elasticsearch ETD cluster.

Acknowledgements

Edward A. Fox

Virginia Tech Computer Science Professor

Our group acknowledges Edward A. Fox for giving us the opportunity to work on this project and supporting our group to stay on track.

Jian Wu

Assistant Professor in the CS Department of Old Dominion University

Our group acknowledges Jian Wu for giving us the opportunity to work on this project and giving us help whenever we encountered problems.

William Ingram

Assistant Dean and IT Services Director of VT University Libraries

Our group acknowledges William Ingram for giving us the opportunity to work on this project and providing us with resources.

Winston Shields

Computer Science Graduate Student of Old Dominion University

Our group acknowledges Winston Shields for providing us with the base code for the website as well as helping Sa Hyun Min with running the website.

Bipasha Banerjee

Ph.D. student and Graduate Research Assistant at Virginia Tech

Our group acknowledges Bipasha Banerjee for providing us with guidance in dealing with the extraction of figures and the creation of our extraction pipeline, as well as providing a remote machine for hosting.

References

[1] Allenai, “Allenai/PDFFIGURES2: Given a scholarly pdf, extract figures, tables, captions, and section titles.,” *GitHub*, 16-Nov-2021. [Online]. Available: https://github.com/allenai/pdffigures2. [Accessed: 10-May-2022].

[2] “Anaconda distribution,” *Anaconda*, 2022. [Online]. Available: https://www.anaconda.com/products/distribution. [Accessed: 10-May-2022].

[3] deepset, “Haystack,” *Haystack Docs*, 2020. [Online]. Available: https://haystack.deepset.ai/overview/intro. [Accessed: 10-May-2022].

[4] Elastic, “What is Elasticsearch?,” *Elastic*, 2022. [Online]. Available: https://www.elastic.co/what-is/elasticsearch. [Accessed: 10-May-2022].

[5] “How to navigate to C Drive in bash on WSL-ubuntu?,” *Ask Ubuntu*, Aug-2017. [Online]. Available: https://askubuntu.com/questions/943006/how-to-navigate-to-c-drive-in-bash-on-wsl-ubuntu. [Accessed: 10-May-2022].

[6] “How to run PHP files on my computer,” *Stack Overflow*, Jan-2012. [Online]. Available: https://stackoverflow.com/questions/8580273/how-to-run-php-files-on-my-computer. [Accessed: 10-May-2022].

[7] J. Reynosa and S. H. Min, “CS4624_Figure_Extraction_Website,” *GitHub*, 26-Apr-2022. [Online]. Available: https://github.com/JRReynosa/CS4624_Figure_Extraction_Website. [Accessed: 10-May-2022].

[8] J. Viluan , “PHP file upload error tmp_name is empty,” *Stack Overflow*, Jun-2015. [Online]. Available: https://stackoverflow.com/questions/30358737/php-file-upload-error-tmp-name-is-empty. [Accessed: 10-May-2022].

[9] J. Wu, “Jian Wu, Old Dominion University: CiteSeerX,” *Jian Wu, Old Dominion University | CiteSeerX*, 2018. [Online]. Available: https://www.cs.odu.edu/~jwu/. [Accessed: 10-May-2022].

[10] Jsvine, “Jsvine/pdfplumber: Plumb a PDF for detailed information about each char, rectangle, line, et cetera - and easily extract text and tables.,” *GitHub*, 06-Mar-2016. [Online]. Available: https://github.com/jsvine/pdfplumber. [Accessed: 10-May-2022].

[11] Lamps-Lab, “Lamps-Lab/etdui,” *GitHub*, 08-May-2021. [Online]. Available: https://github.com/lamps-lab/etdui. [Accessed: 10-May-2022].

[12] *sbt*, 27-May-2014. [Online]. Available: https://www.scala-sbt.org/download.html. [Accessed: 10-May-2022].

[13] Virginia Tech, “William A. Ingram,” *https://experts.vt.edu/5675-william-a-ingram*. [Online]. Available: https://experts.vt.edu/5675-william-a-ingram. [Accessed: 10-May-2022].

[14] “Where can I find php.ini?,” *Stack Overflow*, Jan-2012. [Online]. Available: https://stackoverflow.com/questions/8684609/where-can-i-find-php-ini. [Accessed: 10-May-2022].