College of Engineering

COMP 491 – Computer Engineering Design Project

Final Report

COMPUTER VISION BASED WEB SCRAPER

Participant information:

Batu Helvacıoğlu 69009
Berkut Gökmen 68959
Can Gözpınar 68965
Koray Tecimer 69387

Project Advisor(s)
Barış Akgün

Fall 2021

Table of Contents

I. Abstract

Data is more important than ever. Many companies, workplaces, and users need data in order to accomplish their goals. Web scrapers are an important part of this because they are required to collect a big mass of data from the Internet. Conventional web scrapers are hard to use, maintain and require technical expertise. This results, because the web scraper is built looking inside the webpage, HTML. Elements are extracted by looking at the code of a webpage which takes away robustness. This paper addresses using computer vision for web scraping which can significantly decrease the work needed for web scraping. Using computer vision each important element from a webpage is extracted using object detectors and presented to the user. This paper aims to provide a solution to easy web scraping and the results indicate there can be change in how we and companies use web scrapers.

II. Introduction

Conventional web scrapers work by extracting information from the HTML code that is used to render web pages in browsers. This approach requires technical expertise to be able to find and extract the correct information from HTML elements. It is also susceptible to changes in the HTML code of the web page. If the website changes some element or functionality of a page the new HTML code would not be compatible with the web scraper that was configured to get the old HTML elements. This would force the user to modify the web scraper often by coding a new script or changing the existing script accordingly. One final problem with conventional web scraping approaches is that some websites might try to hide information by using different methods such as showing information with images instead of conventional HTML elements. Shopping sites can be an example for this as some of them hide the price information with images rather than putting it as text into a HTML element. We aimed to reduce the technical expertise and effort required to scrape web pages by creating a robust tool that can be used to scrape shopping, forum and news sites without the user needing to code any scripts or do any configuration. We utilized computer vision models to extract information from web pages. These machine learning models require no configuration to be used with different web pages since they can generalize to a type of website such as shopping rather than being compatible with a single shopping site.

Web scraping is a form of data extraction which is performed on websites. Conventionally, these web scraping libraries perform by parsing the rendered html content which can be used to target certain components and interact with the information that it holds. This way of web scraping relies on the front-end development understanding of the developer and requires css media queries to extract information. Moreover, some advanced web scraping libraries such as jsoup would allow the rendering of javascript code on the target website to extend the capabilities of the web scraper. Even though this way of extracting data works, it bears its own drawbacks. For example, these css queries depend on the html content of the target website. When the html content is altered or naming of the attributes has changed, the css queries break which leaves the conventional web scraper to not work. These changes to websites could happen if the website is going under new design change, or sometimes websites which do not want to be scraped using web crawlers make changes to their html content in order to break other parties' web scrapers. This vulnerability to alterations and needing to have web scraper developers to have an understanding of front-end development technologies has their downsides. This makes maintaining web scraper code harder which could be improved by having a computer vision based scraper that targets elements intuitively like human users does since certain components of interest share many aspects among different implementations of the same type of sites. Upon successful embedding of this way of scraping websites, users can easily scrape websites for certain information with almost no software development knowledge required in a robust way. Our product could be provided as an API to other websites such as akakce.com which is a very popular website that allows one to compare the status of certain items from various websites using conventional web scraping techniques. With the embedding of our product as an API, one could improve the problem of developing web site specific web scrapers which requires heavy maintenance.

III. System Design

We started with writing a URL scraper for shopping, forum and news sites to later train our object detector model. The script allowed us to get URLs of thousands of web pages with a single script. Then we fed URLs to our screenshot script to take the screenshot of every page. These screenshots later were going to be labeled and be used for training the YOLOv5 object detector model.

For labeling data we used an open source program called LabelImg. [2] LabelImg allowed us to create bounding box labels in Pascal-voc or YOLO format. We chose to use the Pascal-voc format since we had access to Python scripts to change it to other label formats if we wanted to use models other than YOLOv5. [3] We examined web pages to determine classes that we were going to label. We decided on about 20 classes for shopping sites and about 10 for news and forum sites, which can be found in the appendix. These classes consisted of some buttons for navigation purposes as well as classes containing information that will be directly served to the user. Manually labeling bounding boxes for all these classes required too much time to be viable. Therefore we decided to create an auto labeller that would utilize conventional web scraping techniques to take web page screenshots and create bounding box labels for these screenshots. We used Puppeteer which is a Node library that can be used to scrape web pages using headless Chromium instances. Using the ‘web page URLs JSON’ that we created previously with our URL scraper script, we visited web pages to take screenshots and get bounding box coordinates from HTML div elements in the site. It is simple to change the web site that is getting scraped by just changing the website name variable in the script and providing the HTML element class names to the corresponding array thanks to our modular approach. However the saved labels are not in YOLO txt format since Puppeteer cannot access the width and height of the full page screenshot that it takes. To solve this problem we created a separate Python script that converts the labels to YOLO txt format by getting the width and height needed for scaling the labels directly from the saved screenshots.

We used YOLOv5 object detectors for our computer vision models. [4] We wanted to make sure that prediction on website screenshots would not bottleneck our system in terms of speed Therefore we chose the YOLO family of models since they have fast inference time and accuracy balance compared to other object detectors. We created separate custom datasets of

bounding box labeled web page screenshots for our 3 supported website types; shopping, news and forum. We used transfer learning to be able to get better results, specifically we used YOLOv5 small trained on COCO dataset. Pretrained models can detect low level features such as edges and corners easier since they were already trained which allows us to cut down on training time.

Inferring the type of website from a given URL is a quality of life feature we added for users. Our website can function by either selecting the type of the website prior to giving the URL to the web page or by providing just the URL with the “Infer the type” option selected. We take the screenshot of the web page just like the other requests but before sending it to the corresponding object detector, we use an image classification model to determine the website type. For the website classifier we experimented with pretrained Resnet152, Resnet34 and Resnet18 models pre trained on ImageNet dataset. We chose to continue with Resnet18 since it is easier to train due its shallower architecture and does not lose a significant amount of performance in our use case compared to the deeper models. We used Roboflow for formatting our dataset consisting of the screenshots we took. [5] Another experimentation we did with the website classifier was to use the Fastai framework and versus directly using Pytorch. Fastai caused package conflicts when we tried to use it for inference, therefore we decided to continue with Pytorch directly without the Fastai framework.

As an added functionality we do sentiment analysis on product reviews from scraped shopping site pages. We utilized Google’s Bidirectional Encoder Representation from Transformers (BERT) language model for this task. [6] BERT uses transformers to create word embeddings that can represent words with a 768 dimensional vector. This representation captures the meaning and relations between words. These embeddings can then be used for other tasks such as sentiment analysis by feeding them into simple classification networks consisting of fully connected layers. For the BERT model we used bert-base-turkish-sentiment-cased a BERT model fine tuned on Turkish sentiment analysis datasets. These datasets were Beyazperde.com movie reviews, a product review dataset and a Twitter dataset created by Boğaziçi University. We further trained this model on another product review dataset but did not achieve significant enough performance gains. During inference we pass all the reviews we extracted from a product page to the sentiment analysis model and return positive, negative and neutral review

counts.

We used Node.js and Express.js for our backend. Node.js was used to develop main functionalities and Express.js was used to create our server. In our backend server we had two endpoints. First endpoint was for inferring the type of the website. User sends a request to this endpoint with the URL of the page. Then the backend runs the machine learning model for inferring the type on this URL. Result of the model is returned to the frontend. With the type that we sent, the frontend sends us a second request with URL and the type of the website to our second endpoint which is for our main functionality: web scraping using computer vision. We divided our service into 5 parts depending on the functionality. Screenshot, Data Scraping, OCR, Data Export, and AI. Each service had their own script and was controlled by a master script called Service. In the screenshot part we used puppeteer which is a web crawler. This script took screenshots of the whole website given the URL. In the data scraping part, we clipped the screenshot image with an image library called sharp according to the location data that was given to us by our AI part. This resulted in individual images for every label that AI detected. After that, we filtered those images and fed the ones that we wanted a text version of to the OCR. OCR created text versions of these images and sent them to our sentiment analysis algorithm. We used Tesseract which is an OCR engine made by Google. We changed its data points to better detect turkish. In the end, data export combined the texts from ocr and the images from the AI as a JSON file and sent them to the frontend using Express.js.

For the development of the user interface we have decided to develop a web browser based user interface. This way we had a user interface which works platform independent. In other words, regardless of the operating system, the choice of web browser or the type of the device, the user would most likely be able to use our user interface to interact with our product. For the front-end development we decided to use the Angular 2+ framework. We benefited from Angular's component based design support. This way we were able to reuse our code and reduce the redundancy in the code written by being able to convert relevant elements of the user interface into related components. Moreover, Angular has a built-in typescript support. Typescript is a programming language that gets compiled into javascript so that it can run on web browsers. Unlike javascript, typescript supports object oriented programming and adds static typing support. Having object oriented programming support helped us with

implementing design patterns and it is much more like java language which we are more accustomed to coding in. We have leveraged the benefits of using popular javascript libraries in many places but two of them were essential namely, angular material and bootstrap 5. We have used Angular material for the component designs. In other words, we did not go through the process of designing and implementing user interface elements but instead, we have used angular materials components. These components were good looking and user friendly so they met our expectations. Furthermore, since web browsers are supported by many different devices with drastically different screen sizes, we needed our user interface to adapt its design to the screen size of the device dynamically. For this purpose instead of using media queries in css in a conventional way, we utilized bootstrap 5 library. Bootstrap 5 is the fifth release of the bootstrap framework which is open sourced by Twitter. Bootstrap is very famous in the industry and has many built in functionalities that makes designing responsive (adaptive to various screen sizes) web pages much easier. The user interface that we have built mainly serves the purpose of taking the url of the web page of interest as a user input. Then queries the back end services with the corresponding body and parses the returned response which has the scraped information. Later, a table with the corresponding columns with certain modal windows are shown to the user to investigate the results of the scraping that got returned by the back end services.

IV. Analysis and Results

We managed to train object detectors that could predict the classes that contained information accurately. However our models struggled with detecting some of the classes that we used for navigation purposes. Fig 1 is the confusion matrix of a shopping site object detector that we trained and tested on Trendyol.com. It can be seen that most of the classes that actually contain information were detected by our model. Most of the misclassifications were between classes similar to each other such as selected photo and photo. The difference between selected photo and photo classes is that there is an outline around the selected photo class to indicate it is the one being viewed in large format. Rest of the errors that the model made were primarily caused by not being able to detect navigation buttons such as the reviews button from the background. We believe our model struggled with these classes since Trendyol's buttons are

the same color as the web page background.

We have successfully developed a responsive (dynamically adapts to various screen sizes) and platform independent user interface that can be used for interacting with our product in a user friendly manner. The user interface is able to take user input, switch between supported modes and query the back end services for the corresponding scraping information.

Initially we created our backend using Spring boot but we decided that Spring boot was too verbose and Node.js was easier to use when it comes to using it with Angular. Our backend works slow and queries take time to be satisfied. It has a lot of moving parts in terms of working with different machine learning models. All functionalities are satisfied that come from the user.

We have developed an auto labeler which we used to scrape websites for data to train our machine learning models with. Our auto-labeler worked by scraping websites for certain products given a product type and saves the url's scraped into a json object. This object is later used by the continuity of the auto labeler script to take screenshots of the images and provide YOLOv5 compatible annotation files for every screenshot taken which can be directly used for training the YOLOv5 object detectors. This way we have tackled one of the most important problems in training machine learning models namely, labeled data acquisition for supervised learning tasks. Moreover, in comparison to the time that went into developing these scripts, we have acquired more labeled data than we could have acquired if we were to gather these using manual labor. This way, we managed to save our precious time for development instead of spending it for labeling the data.

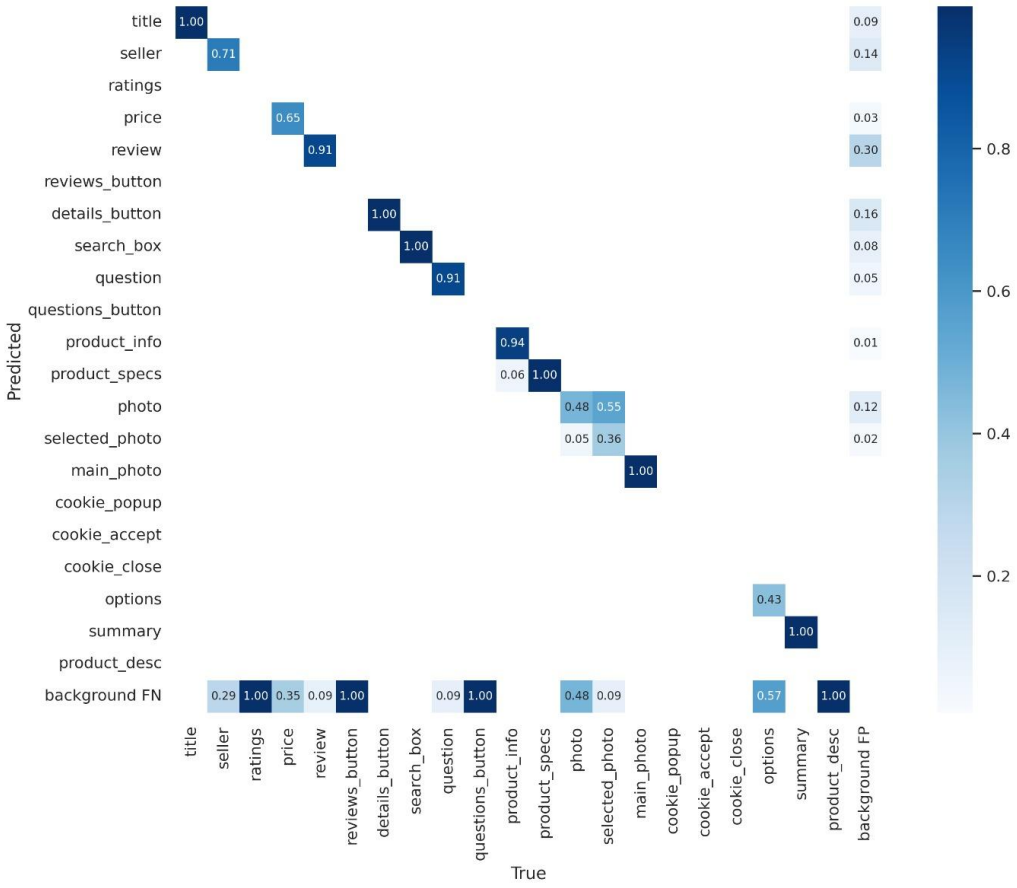

Fig. 1 Shopping Site Object Detector Confusion Matrix

Fig 2 shows the confusion matrix of our news site object detector trained and tested on Sozcu.com. It can be seen again that navigation classes such as cookies accept button were not detected accurately. Classes that contain actual information were detected with high accuracy except the subtitle class. This was due to the cookie popup blocking the subtitle in Sozcu.

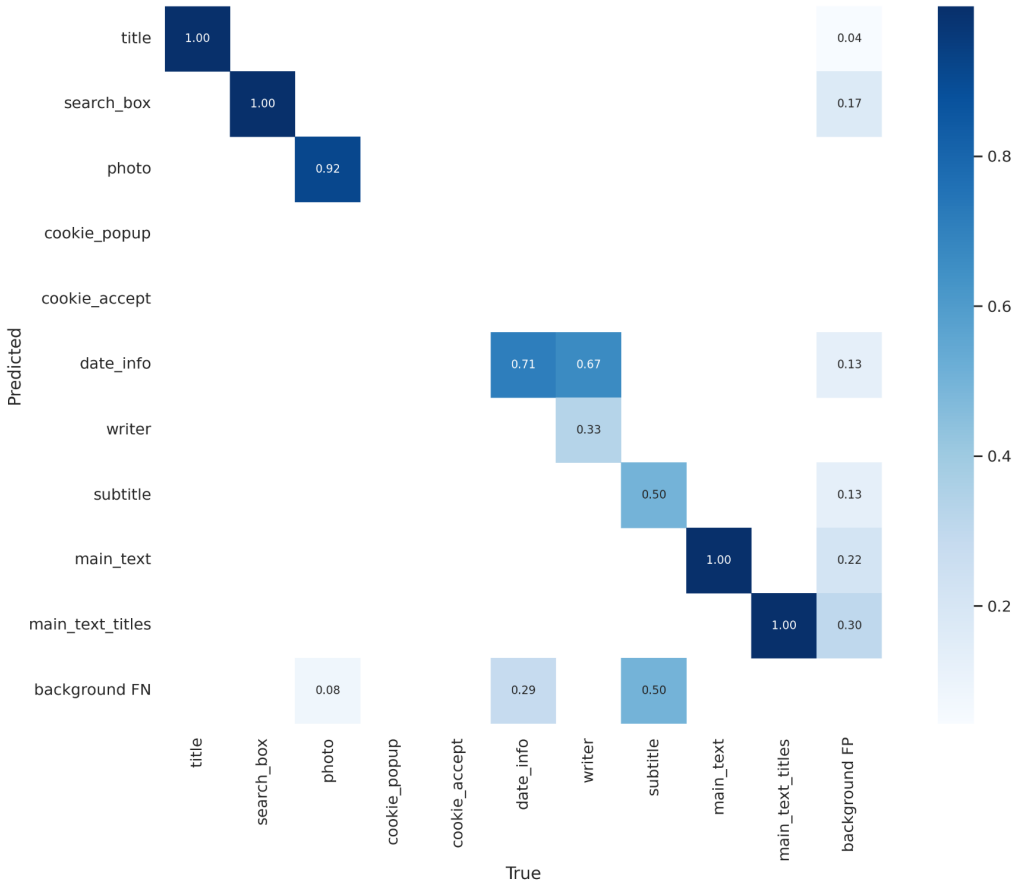

Fig. 2 News Site Object Detector Confusion Matrix

Finally figure 3 shows the confusion matrix of our forum site object detector trained and tested on Eksisozluk.com. All of the classes that contained information were detected successfully. There are no answer classes predicted since our screenshots for eskisozluk.com did not contain answers of the posts.

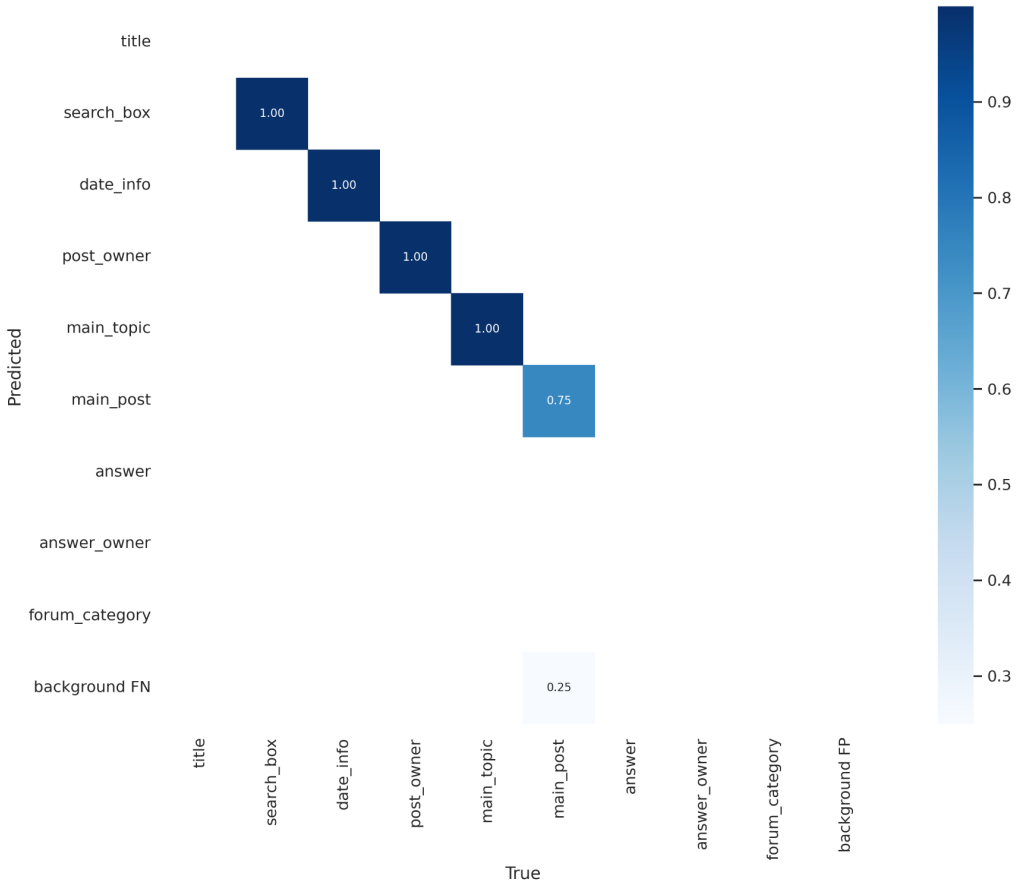

Fig 3. Forum Site Object Detector Confusion Matrix

Even though the object detectors showed great potential to generalize we could not test their generalization properly since we could not train models with data from more sites due to a technical difficulty with Google Cloud. However we observed that the news stie object detector that was trained on Sozcu managed to detect some of the classes in Hurriyet.com. Our sentiment analysis model got around 70% accuracy on the dataset that we trained it on. Website classifier that we used to infer the website type got around 90% accuracy in our validation set but struggled with detecting forum sites in practice. It performed better when classifying shopping and news sites.

The auto-labeller that we created performs well and can create accurate labeled data. It has some problems with disabling cookies by loading Chrome extensions since extensions can only be used in headful mode of Chrome. When disabling cookies auto-labeller slows down because it can no longer function in headless browser mode. We encountered some problems

where the screenshot was taken before all the elements in the site were loaded. There were also some instances where sites blocked the auto-labeller because they detected unusual activity. Apart from these issues, the auto-labeller script worked well and is able to produce datasets with minimal manual effort of cleaning up the screenshots of unloaded and blocked sites.

V. Conclusion

Results for the shopping sites are good but forum sites and news sites have some defects due to the fact that news and forum sites were not trained with sufficient amounts of data. OCR results are inconsistent as it can yield perfect image to text translation and sometimes texts full of misspelled words.

As for the improvements, we tried to make our screenshot process concurrent but failed to do so. We could make the whole process concurrent. For implementing concurrency, we need to tie every part of our service together without doing any file system IO. This can be done using sockets between scripts. Puppeteer requires further development for concurrency. We can create crawler clusters and take screenshots using these clusters.

We can migrate the OCR engine to python to better tweak the parameters and get better results. It is also recommended to do preprocessing before passing the images to OCR which can be done easily in Python.

We can add navigation after the web scraping is done as we already label the buttons.

It is possible to further improve the object detectors by training them with a dataset that contains a larger variety of websites. All of our machine learning models can perform better if they are trained for longer epochs.

VI. References

[1] E. C. Dallmeier, "Computer Vision-based Web Scraping for Internet Forums," *2021 7th International Conference on Optimization and Applications (ICOA)*, 2021, pp. 1-5, doi: 10.1109/ICOA51614.2021.9442634.

[2] Tzutalin (2021) LabelImg (Version 1.8.6) [Source code]. https://github.com/tzutalin/labelImg.

[3] A., “Convert pascal VOC dataset to Yolo Format,” Gist, 2020. [Online]. Available: https://gist.github.com/Amir22010/a99f18ca19112bc7db0872a36a03a1ec. [Accessed: 16-Jan-2022].

[4] G. Jocher, *YOLOv5 Documentation*, 2020. [Online]. Available: https://docs.ultralytics.com/. [Accessed: 16-Jan-2022].

[5] B. Dwyer, “Getting started with Roboflow,” *Roboflow Blog*, 29-Apr-2021. [Online]. Available: https://blog.roboflow.com/getting-started-with-roboflow/. [Accessed: 16-Jan-2022].

[6] S. Yıldırım, *savasy/bert-base-turkish-sentiment-cased · Hugging Face*. [Online]. Available: https://huggingface.co/savasy/bert-base-turkish-sentiment-cased. [Accessed: 14-Oct-2021].

VII. Appendix

Classes for shopping sites: 'title', 'seller', 'ratings', 'price', 'review', 'reviews_button', 'details_button', 'search_box', 'question', 'questions_button','product_info', 'product_specs', 'photo', 'selected_photo', 'main_photo', 'cookie_popup', 'cookie_accept', 'cookie_close', 'options', 'summary','product_desc'.

Classes for news sites: "title", "search_box", "photo", "cookie_popup", "cookie_accept", "date_info", "writer", "subtitle", "main_text", "main_text_titles".

Classes for forum sites: "title", "search_box", "date_info","post_owner", "main_topic", "main_post", "answer", "answer_owner", "forum_category".