JSLabeling: A Novel Approach to Labeling the Web

Christopher Chen
Computer Science, NYUAD
pcc386@nyu.edu

Advised by: Yasir Zaki

ABSTRACT

The Internet and its web pages have become crucial components in today's world. Web pages in particular, serve a critical role in providing a platform for communication, interaction and the dissemination of knowledge. With advancements in the field of web development, the web has become increasingly complex, making web analysis more difficult. Additionally, this increased complexity comes at a cost; since not everybody has access to new technologies, the Internet becomes less accessible to these people. This is creating a digital divide.

The aim of my Capstone is to label web elements on the Internet and to build models that would identify and differentiate these components. The end goal of using this research is to reduce website loading times and therefore make web pages more accessible. I have conducted a user study where users will identify and label various web elements on a web page. To accomplish this, I built a tool that allows users to interact with a web page and then select and label its web components. With the data gathered from this study, I spent the semester training machine learning models that would recognize and match different web elements. Further evaluation will be conducted to ensure the accuracy and validity of the trained models. These models then can be further used in a myriad of ways such as recognizing and removing unnecessary elements.

KEYWORDS

Computer Vision, Internet accessibility, machine learning, web page analysis, web development, web page segmentation, website labeling

This report is submitted to NYUAD's capstone repository in fulfillment of NYUAD's Computer Science major graduation requirements.

جامعة نيويورك أبوظبي
NYU ABU DHABI

Capstone Project 2, Spring 2021, Abu Dhabi, UAE

Reference Format:
Christopher Chen. 2021. JSLabeling: A Novel Approach to Labeling the Web. In *NYUAD Capstone Project 2 Reports, Spring 2021, Abu Dhabi, UAE*. 8 pages.

1 INTRODUCTION

The Internet and its web pages have quickly become an integral platform for discourse and the dissemination of information. It is a rapidly advancing technology that will only continue to improve and increase in complexity. However, the continual advancements come at a cost: an increased amount of resources needed. In *Constructing Novel Block Layouts for Webpage Analysis*, Jiang et al. [3] point out that web technologies are becoming more dynamic and complex, requiring more resources in addition to presenting more obstacles for web page analysis. When a web page is loaded by the browser, it creates a Document Object Model (DOM) of the page. The DOM is a tree structure wherein each node is a specific DOM element that represents an element on the web page. This DOM structure results in a segmentation of the web page where the page is divided into different functional regions such as the header, footer, search bar, etc. Proper web segmentation is helpful in reconstructing web pages for different devices. This also includes the possibility of reducing components that would require significant amount of resources to parse.

However, the introduction of new front end frameworks, such as React, makes it difficult for web segmentation. The process of web segmentation looks at the correlation between similarly grouped elements. Content that are closely positioned or logically related would be grouped closely on the DOM model. In modern frameworks, this is not the case; the structure of web pages are more flexible and ordered inconsistently. Additionally, web page designs are becoming more sporadic with a more fluid structure, making web page analysis much more difficult.

JSLabeling provides a method for web page analysis by breaking down web pages into smaller web elements. It takes a higher level approach by looking at web pages from a visual point of view instead of code. Users are tasked to directly identify, select, and label web elements directly on the web

page. Thus, web pages are partitioned into its components. These labeled elements will then be used to create machine learning models that are able to discern.

Another issue stemming from increased web complexity is the significantly increased web page load times. In *Web-Page Complexity and Optimization Mechanism to Reduce Web-Page Load Time* [6], Omkar Sawant and Sachin Godse explains that longer web page load times deteriorates the user's experience and satisfaction, resulting in users clicking away. This is especially problematic for users who do not have the necessary technology to access the increasingly complex web pages and creates a digital divide of Internet accessibility. With the machine learning models built from JSLabeling, we will target this issue by fine tuning our models to recognize elements that are unnecessary and then remove them.

2 RELATED WORK

2.1 Web Page Analysis

The first half of my Capstone, as detailed in my Capstone report, focused on labeling web components. There have been several works that target web page analysis and reducing page load times.

Jiang et al. [3] propose a two-step web segmentation method that groups visual, logic, and semantic features of the contents of a web page. It first creates the general layout of a web page and then measures the similarity of different web elements. Afterwards, it clusters similar elements into larger content blocks for web page segmentation.

Andrés Sanoja and Stéphane Gançarski propose a hybrid approach to web page segmentation called Block-o-Matic [5]. It entails three phases: analysis, understanding and reconstruction. Their first phase obtains the DOM tree from the browser and then generates a content structure. In the next phase, it maps the content structure into the logical structure of the web page by categorizing DOM elements. Lastly it reconstructs a document tree with the logical structure generated.

Another method by Brian Burg, Andrew J. Ko, and Michael D. Ernst [1] directly locates the code that creates the elements and implements interactive behaviors. Their tool Scry looks at the DOM and Cascade Style Sheets (CSS) of a selected element and takes a screenshot of the element. It can be used to look at state changes of elements and output the lines of code that were responsible for the changes.

In terms of web page complexity, Omkar Sawant and Sachin Godse describe a tool that measures the complexity of web pages [6]. After the tool has analyzed a page, it suggests different optimizations that would reduce web page complexity and page load times. The optimizations include, moving scripts to footer, loading JavaScript from Google libraries, removing query strings, lazy loading images to improve speed, removing extra font styles, and loading CSS asynchronously.

Another tool, JSCleaner [2], attempts to reduce web page complexity by transforming web pages to simpler versions. It extracts inline and external JavaScript elements and categorizes them into *critical*, *replaceable*, and *non-critical*. It then removes all non-critical elements and replaces JavaScript elements with HTML elements while keeping the critical components.

2.2 Machine Learning

The second half of my Capstone report focuses on training machine learning models to identify and label web elements. We decided to devote our time into the Contrastive Language-Image Pre-Training (CLIP) neural network by OpenAI that was recently launched earlier this year.

In the new CLIP model [4], Radford et al. created a neural network that is trained on image text pairs. Given an image, it uses natural language processing to find a caption that fits the image. It is largely built upon zero-shot transfer, natural language supervision, and multimodal learning. Oftentimes, images are paired with text on the web (captions). CLIP models take into consideration that there are a wide variety of visual concepts in images and associate them with their captions. This neural network is intended to be used in a zero-shot manner, thus the data it uses vary greatly. As a result, CLIP models are general and can be applied to various different types of data sets.

3 METHODOLOGY

3.1 Web Labeling Tool

JSLabeling includes two features worth noting: an intuitive user interface and an accurate method in gathering data on web elements.

The user interface of JSLabeling is composed of an iFrame on the left side of the users screen that showcases a web page and a p5.js sketch on the right side that contains an image of the same web page with labeling buttons on the top (refer to figure 1). The workflow of the tool is simple and straightforward: the user interacts with the web page on the left (iFrame) to determine the various functionalities of the web elements. After the user has familiarized themselves with the web page, the users are tasked to label the web elements on the right side (the image on the p5.js canvas). This interface allows users to reference back to any unknown elements by simply interacting with the iFrame on the left.

The process to label a web element is simple: the user must click and drag an outline of the element, much like the marquee tool in Adobe Photoshop or Illustrator. This will generate a red rectangle that references the current area

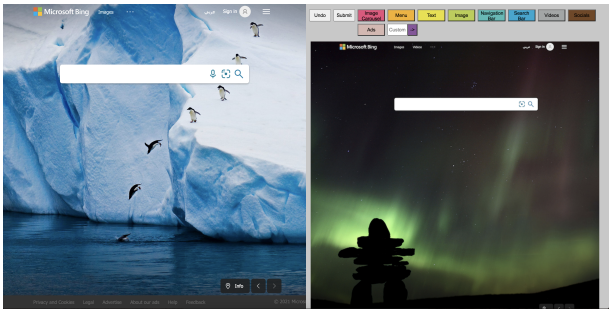

Figure 1: An image of the user interface

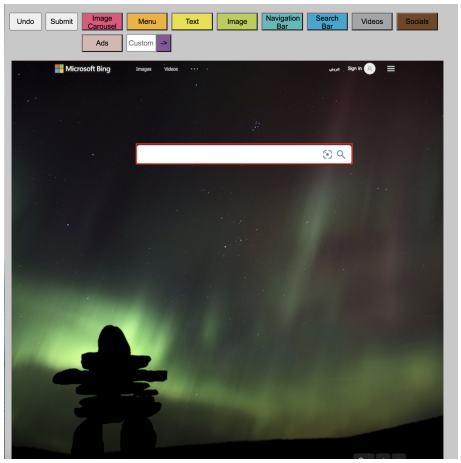

Figure 2: An image of the search bar selected

that is selected (refer to figure 20). Upon mouse release, the rectangle will stay in place and the user will be able to label the selected element by pressing one of the buttons on top. The list of web elements are as follows: image carousel, menu, text, image, navigation bar, search bar, videos, socials, ads, and a custom tag. In cases where the above mentioned labels do not fully describe a web element, users are able to generate their own tag with the custom tag. Additionally, all of the label buttons are color coded; when the selected area is labeled by pressing on one of the label buttons, the rectangle will take on the same color as the label selected (refer to figure 3). Additionally, the x-coordinates, y-coordinates, width, height, website, and label are stored. The coordinates are used later on to crop a higher quality image of the same web page.

Before the labels are used for to train our machine learning models, the labels have to be filtered and cropped. Using the coordinates from the labeling phase, a Python program will crop the labels from a higher resolution image of the web page. The labels will be sorted in separate folders that reference its label type and used to train machine learning

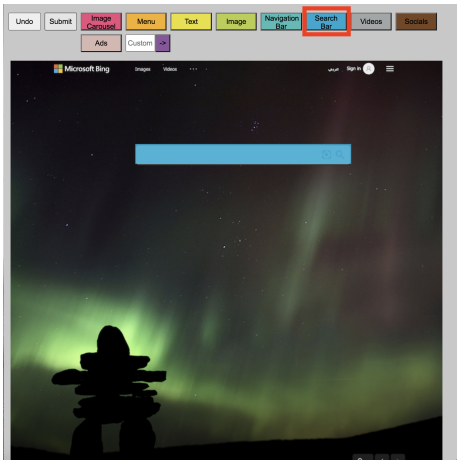

Figure 3: An image of the search bar labeled

models that will at first identify and recognize different web elements.

3.2 Machine Learning Models

For the semester, most of my effort has been focusing on training machine learning models to recognize and label different web page elements. We decided to focus on two different machine learning models: OpenCV's cascade classifiers and OpenAI's Contrastive Language–Image Pre-training neural network.

Capstone 1 was dedicated to training OpenCV's Haar cascade classifiers to identify and label web elements. Cascade classifiers are trained by using positive and negative images. Positive images are images of what we want the model to identify and negative images are anything else. The Haar cascade classifier first creates Haar features that are calculations performed on rectangular regions at specific locations of a detection window. Essentially, it calculates the sum of a pixel density within a given rectangular region and then compares it against the sum of different regions to find the difference. Calculating the sum of every single pixel of the image and comparing it with other regions is computationally heavy, thus cascade classifiers utilize integral images to reduce the load. Integral images speed up these calculations by grouping pixels into a small rectangle and creating an array reference to each of these rectangles. Then, it uses Adaboost training to choose the best features by using a combination of "weak classifiers" to create a "strong classifier." Adaboost uses a sliding window over the input image and computes Haar features on each subsection of the image (integral image), creating weak learners. Cascade classifiers utilize a large amount of weak learners to create strong learners. Through a process called cascading classifiers, weak learners are trained using boosting and then used to predict if an object is found.

Figure 4: Cascade Classifier 10 Steps, Double Negative Sample Size

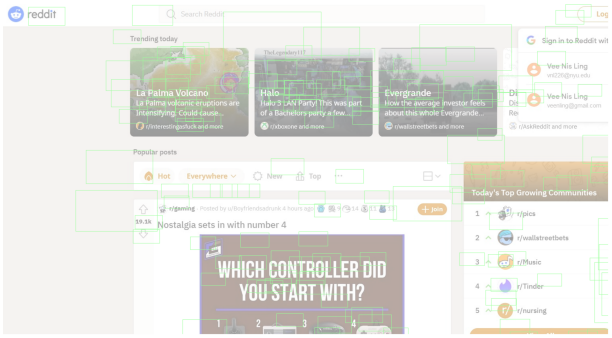

Figure 5: Cascade Classifier 10 Steps, Half Negative Sample Size

This process is repeated for multiple stages until the model is finished training.

Traditionally, cascade classifiers were used for identifying faces, but we decided to see if it would be able to detect web elements. Using the data gathered from the web labeling user study, I trained a model to identify search bars. To test if my model worked, I would use the screenshot of different websites and see if the model would be able to locate the search bar located on the page. The results I obtained were subpar as the model was never able to only locate the search bar. The first model I trained had the following parameters: trained for 10 steps, a detection window of 60px by 30px, and double the negative images to positive images (see figure 4).

After seeing the initial results, I attempted to play around with the parameters in hopes of better results. I tried using half the amount of negative samples (figure 5), increasing the number of steps (figure 6), the hit rate and false alarm rate (figure 7), and detection window size (figure 8).

Despite changing the parameters, the model was unable to precisely locate the search bar. A possible explanation for this is the small sample size that was used. Additionally, it could be that cascade classifiers have a hard time differentiating between different web elements.

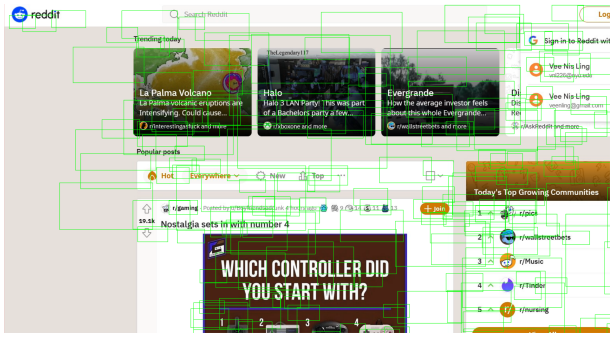

Figure 6: Cascade Classifier 12 Steps, Half Negative Sample Size

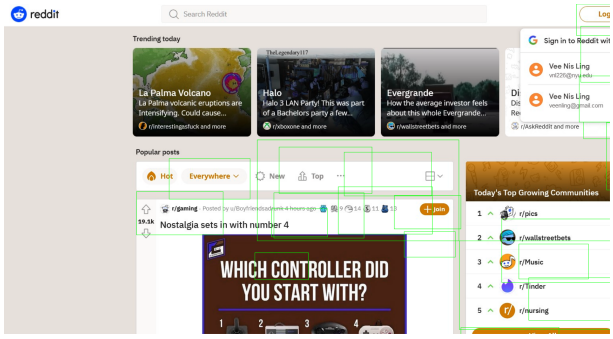

Figure 7: Cascade Classifier 12 Steps, Double Negative Sample Size, False Alarm Rate: 0.3, Hit Rate: 0.999

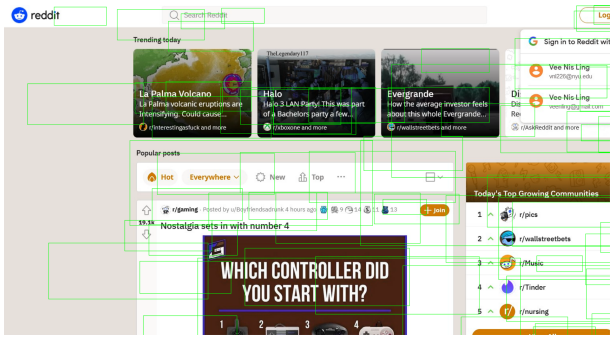

Figure 8: Cascade Classifier 10 Steps, Double Negative Sample Size, Detection Window: 100px by 35px

After training the cascade classifier models, we attempted to use OpenAI's CLIP models. CLIP is unique in that it looks at the relationship between a whole sentence and the image it describes; it is trained on full sentences instead of single classes. The idea is that by looking at whole sentences, the model can learn more things and identify some pattern between images and texts. As a result, it can identify abstract

concepts within an image through the semantics of the attached caption in addition to differentiating between simple objects. Although CLIP can serve as a zero-shot image classifier, it can also serve as a classifier by itself. We understood that our data size was insufficient to train an accurate model, thus we opted for the transfer learning approach. By reusing a pre-trained model, training CLIP with our web image data set, we can obtain a model that is customized more towards our needs of identify different web elements. We utilized our CLIP model in two ways: given a query, return the matching images and given an image, return the prediction of its category.

Before we train our CLIP model, we have to encode both our images and its describing text. We used the DistilBERT model from the HuggingFace library as our text encoder. It tokenizes the sentences with DistilBERT tokenizer and then feeds the token ids and the attention masks to DistilBERT. Afterwards, we used ResNet50 from PyTorch Image Models library (timm) to encode our images, converting them to a fixed size vector with the size of the model's output channels. Once we have converted both the text and the images, we project them onto the same space and compute the loss by finding the dot product between the two vectors. By doing this, we want the model learn "similar representations (vectors)" for a given image and the caption describing it. At the end, all our vector pairs are converted a matrix and fed into the CLIP model to train.

We trained our model with the following web elements: menu bars, search bars, socials, images and text. For each of the elements, I standardized the captions with "An example of a web element." To test the model, I used different captions to query it: "search bar (figure 9)," "an image of a search bar (figure 10)", and "An image of a search bar taken from a web page (figure 11)." This model was also tested on socials (figure 12) and menu bars (figure 13).

From the results gathered, it was evident that the model struggled to identify different web elements. The most accurate was the search bar, and even then, it was not very accurate. Changing parameters and the captions used to train the models did not make much of a difference in the results.

To further fine-tune our model, we shifted into returning the accurate category for a given image. When given an image, a list of probabilities for each category was listed. Figure 14 and figure 15 showcases the probability prediction for an image of a search bar and an image.

Our first model seemed to do well in recognizing search bars, images, and text, but failed to accurately predict and recognize socials and menus. Figure 16 and figure 17 list the probabilities for a menu and socials respectively. For many of the predictions, the model seemed to label menu bars and socials as search bars.

Figure 9: CLIP model queried with "search bar"

Figure 10: CLIP model queried with "an example of a search bar"

Given the nature of CLIP, one way to fine-tune the model was to target the captions themselves. Starting with socials, we realized that the model was unable to build the textual relationship between the text "socials" and an image of socials. Thus, we tried various combinations of captions to see if we could improve the performance. This included breaking the socials into each individual social media such as Facebook, Instragram, etc. and retraining our model. This approach did not improve the accuracy of our model for socials. We also retrained our model using the term "social media" instead of socials and it drastically boosted the performance of our model. The model seemed to already have pre-existing relationships between the image of socials and the caption "social media". Figure 18 shows the difference in predicted

Figure 11: CLIP model queried with "An example of a search bar taken from a web page"

Figure 12: CLIP model queried with "An example of socials"

probabilities for an image of social media. Using the caption "socials" predicted the image to be a search bar while the caption "social media" predicted the image accordingly. Compared to our initial model, we were able to boost the accuracy specifically for socials from 20 percent to 75 percent.

The other category that presented much trouble were menus. Since we are utilizing a pre-trained model, we realized that the term "menu" could represent a food menu or a website menu and that a relationship could have already been built on a different definition. Additionally, on a web page, there are various types of menus: menu bars, hamburger menus, and lines of text. We trained our model

Figure 13: CLIP model queried with "An example of a menu bar"

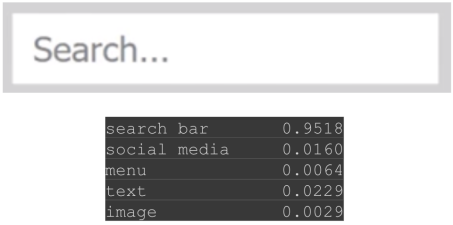

Figure 14: Predicted probabilities for search

Figure 15: Predicted probabilities for image

Figure 16: Predicted probabilities for menu

Figure 17: Predicted probabilities for socials

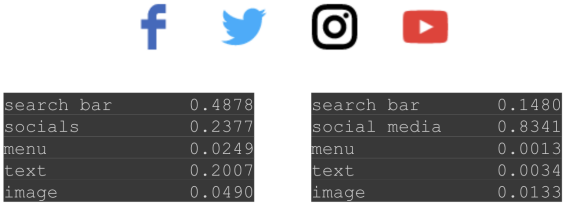

Figure 18: Predicted probabilities for an image of socials with the caption socials and social media

with various different approaches such as using the captions menu or menu bar or separating menus in hamburger menu and menu bars. The captions menu and menu bar did not increase the accuracy of our model. Separating the menu into hamburger menu and menu bars did boost the performance and accuracy of specifically hamburger menus but it significantly decreased the accuracy of other components. We decided to be more specific and retrained our model with the caption "website menu bar" and it significantly increased the accuracy of the model from 11 percent to 81 percent. Figure 19 and 20 shows the predicted probabilities for an image of a menu with the caption "menu" and "website menu bar".

Figure 19: Predicted probabilities for an image of a menu with the captions menu and website menu bar

Figure 20: Predicted probabilities for an image of a menu with the captions menu and website menu bar

4 EVALUATION

JSLabeling was be evaluated by users in two different phases: the process of labeling web pages and the validity of the machine learning models. There was a user study that evaluated the labeling process.

In first user study, participants were tasked to identify, select, and label web page. This entire process was documented. Additionally, the quality of the labels selected (accuracy and precision) will be recorded in the form of a screenshot. The selected labels are also filtered through and cropped before they are used to train models. Lastly, users were surveyed about their overall experience with the tools.

After our machine learning models were built, the validity of the trained models were tested on a test data set. The test data set was composed of around one hundred web component images and the predicted category was generated for each image. The accuracy was then divided into individual categories for each type of web element and an overall accuracy was computed. For predictions that were below the threshold of 50 percent, we decided that those predictions were not conclusive and classified it as categorized. With our trained model, we were able to achieve an overall accuracy rate of 76 percent with individual accuracy of search bars, images, and text reaching 94 percent, 87 percent, and 88 percent respectively. Lastly, we compared our results against the base CLIP model in which we utilized as our starting point. The base model was only able to accurately predict an image 54 percent of the time.

5 PROJECT TIMELINE

This Capstone project was divided into two parts: web element labeling and building machine learning models.

5.1 Web Labeling

The first semester mostly dealt with gathering data from web elements. To achieve this, we designed a tool that allowed users to identify, select, and label different web elements on web pages.

(1) **Design Web Labeling Tool (2 weeks):** The first three weeks will be spent specifying the requirements for the tool along with the user interface.
(2) **Build Web Labeling Tool (3 Weeks):** The tool will be created using HTML/CSS/JavaScript/PHP, iFrames, and p5.js. It will be connected to a MySQL database and then hosted on the server.
(3) **Testing (2 Week):** After the tool has been deployed, it will be tested thoroughly to fix potential bugs or errors.
(4) **User Study (2 weeks):** A user study will be conducted tasking users to label various web elements.
(5) **Data Analysis (2 weeks):** The first two weeks will be spent analyzing the data gathered from the user study. The data will be filtered through so that it is easily cropped and used for the machine learning models.
(6) **Python Cropping Tool (2 Weeks):** A Python program will be built to read from our SQL database (where all the user study data is stored) and crop web pages based on the parameters that were collected during the user study.

5.2 Machine Learning Models

The second part of my Capstone largely involved the data that has been gathered from the tool and using the data to build machine learning models.

(1) **Machine Learning Models (7 Week):** From the cropped web elements, we will continue training machine learning models, primarily with OpenAI's CLIP models. The focus is on fine-tuning the model for accurate results and testing them on various web components.
(2) **Testing (3 weeks):** The model will be trained on test data sets and compared against the base CLIP model.
(3) **Review and Application (2 weeks):** The results will be reviewed and further applications will be discussed.

6 NEXT STEPS

Our next steps involve utilizing QLUE, a tool that segments the web page, to further enhance our model. Given a screenshot of a web page, QLUE is able to isolate each independent element on the page. There are two applications to this: first we are able to further train our model with this data, or more importantly, utilize QLUE to further label the web.

7 CONCLUSION

The Internet is a fantastic piece of technology that contains numerous amounts of possibilities. However, with each improvement it risks leaving many of its users behind. JSLabeling attempts to remedy this bridging divide by providing an approach to web page segmentation and analysis through labeling web elements. These web elements can then be used to create machine learning models that can help identify and recognize web elements, aiding in web segmentation and analysis. With proper web page segmentation and analysis, we can come up with methods to reduce web page complexity and page load times. This could include fine tuning models to identify and remove unnecessary web elements. The greatest attribute of the Internet is its ability to connect people and thus its imperative that accessibility should not become an issue.

REFERENCES

[1] Brian Burg, Andrew J. Ko, and Michael D. Ernst. 2015. Explaining Visual Changes in Web Interfaces. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software amp; Technology* (Charlotte, NC, USA) *(UIST '15)*. Association for Computing Machinery, New York, NY, USA, 259–268. https://doi.org/10.1145/2807442.2807473
[2] Moumena Chaqfeh, Yasir Zaki, Jacinta Hu, and Lakshmi Subramanian. 2020. JSCleaner: De-Cluttering Mobile Webpages Through JavaScript Cleanup. 763–773. https://doi.org/10.1145/3366423.3380157
[3] Zexun Jiang, Hao Yin, Yulei Wu, Yongqiang Lyu, Geyong Min, and Xu Zhang. 2019. Constructing Novel Block Layouts for Webpage Analysis. *ACM Trans. Internet Technol.* 19, 3, Article 35, 18 pages. https://doi.org/10.1145/3326457
[4] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning Transferable Visual Models From Natural Language Supervision. arXiv:cs.CV/2103.00020
[5] Andrés Sanoja and Stéphane Gançarski. 2014. Block-o-Matic: A web page segmentation framework. In *2014 International Conference on Multimedia Computing and Systems (ICMCS)*. 595–600. https://doi.org/10.1109/ICMCS.2014.6911249
[6] Omkar Sawant and Sachin Godse. 2014. Web-Page Complexity and Optimization Mechanism to Reduce Web-Page Load Timep. *IJCAT Journal Volume 1, Issue 9*, 444–447.