Constructing DOM hierarchy from real-world website screen captures

Thisaranie Kaluarachchi

University of Colombo School of Computing, Colombo 0700, Sri Lanka

---***---

Abstract - *Web UI/UX developers typically devote significant effort to designing and developing a desired website, as the website design is constantly reviewed and modified by end users or stakeholders. This repetition is a time-consuming and labor-intensive step in website development. In this paper, a novel automatic DOM hierarchy construction approach based on computer vision and machine learning is proposed to reduce the overhead of the iterative web UI prototyping process by creating the DOM for a website, which can eventually be translated into various front-end web frameworks. The proposed approach constructs the DOM hierarchy from a screen capture of a real-world website as a website mock-up design artifact. It detects and classifies web GUI elements in a website mock-up design artifact using computer vision techniques and a CNN-driven image classifier. To construct the DOM hierarchy for a website, heuristic algorithms are proposed that arrange classified web GUI elements in a tree structure. An in-depth analysis of the approach reveals that it is capable of (I) accurately detecting and categorizing web GUI elements in a screen capture of a real-world website and (II) generating DOM hierarchies similar to those designed by web developers.*

Key Words: website mock-up design artifact, web GUI elements detection, web GUI elements classification, heuristic algorithm, automatic DOM hierarchy construction

1. INTRODUCTION

The DOM construction is an Application Programming Interface (API) for documents written in HTML and XML markup languages. It is an essential part of front-end web development. It provides a way for programmers to interact with and manipulate the structure of a website. With the help of DOM, web developers can access and modify different parts of a webpage. The final mock-up design sketch may not always accurately represent the look and feel of a final web page that the client expects. If the web UI/UX designer does not refine a mock-up design sketch until it reaches the developers, incorrect assumptions may be made during the iterative prototype development phase. Finally, the sketched prototype of the website may not satisfy the client's expectations. The mock-up design sketch leads to iterations of different copies of the design sketch to fine-tune the prototype until an agreement is reached with the client. Functional GUI code should be iteratively modified with the DOM structure accordingly.

Web UI/UX developers typically devote significant effort to designing and developing a desired website, as the website design is constantly reviewed and modified by end users or stakeholders. This repetition is costly in terms of time and resources. As a result of these challenges, as well as the ever-changing web development trends with front-end code automation, an automatic website generation concept has emerged [1], partially to automate web design and development workflow. Automatic website generation is a versatile solution to minimize the burden on the web design and development process while dramatically minimizing web development costs [2-4] and accidental mistakes [5].

According to a recent literature review on automatic website generation [1], researchers have invented several strategies to automate the UI prototyping aspect in the website development process, such as GUI element classification from visual designs, DOM hierarchy construction, and HTML code generation. However, automating the UI prototyping aspect of the website development process is still a big challenge. This leads to the motivation of this research work, which explains a novel automatic DOM hierarchy construction approach from screen captures of real-world websites to reduce the burden and time-wastage of the iterative web UI prototyping process to some extent.

As such, this research presents and algorithm to construct the DOM of a website from the mock-ups. For the research baseline, three steps are considered to be able to create a functional approach: (I) identify GUI elements to realize a website screen capture, (II) determine web GUI elements and their related attributes, and (III) generate the corresponding DOM hierarchy as the last step. These steps are organized in a pipeline, and each step can be modified independently, facilitating the development and experimentation of different modules. The DOM construction approach presented in this paper was compared with the state-of-the-art webpage generation tools *Sketch2Code* [14] and *pix2code* [2]. Comparing the code generated with the DOM construction is, therefore, valid, and the results indicate that the approach produces DOM hierarchies closer to the target DOM hierarchies than *Sketch2Code* and *pix2code*.

The remainder of this paper is organized as follows: Section 2 conducts a literature review of the current state-of-the-art methods researchers have engaged in to automate the web UI prototyping process. Section 3 focuses on the design science research implementation of the proposed DOM hierarchy construction approach. Section 4 evaluates the approach by structurally comparing constructed DOM hierarchies to actual DOM hierarchies constructed by professional web developers. Section 5 expands on the discussion of results, and finally, Section 6 concludes the research by looking at the limitations and future directions of the proposed DOM hierarchy construction approach.

2. LITERATURE REVIEW

With website development trends changing yearly, the automatic website generation trend assists UI/UX developers in automating aspects of the web UI prototyping process, providing a versatile solution to significantly reduce the burden and time wasted in web design and development processes [1]. Two strategies researchers have invented to automate the web UI prototyping process are considered to motivate the proposed automatic DOM hierarchy construction approach. They are web GUI elements extraction and classification from visual designs [2,3,6-15] and DOM hierarchical construction [2,6,7,11,12,14,15].

2.1 Web GUI Elements Extraction and Classification

The related works show recent methods of extracting and classifying web GUI elements via computer vision (CV) and machine learning techniques. Classical CV techniques detect and classify the positions, sizes, and types of web GUI elements from visual designs. The studies [6,7] utilize various CV techniques, such as contour detection, edge detection, colour filtering, and line detection, to robustly classify web GUI elements, such as images, texts, titles, buttons, menus, inputs, etc., in an image. CV techniques face the problem of being extended with new GUI elements, which is challenging and time-consuming.

With the advancement in machine learning, the emergence of new families of machine learning methods was required. Lately, the Convolutional Neural Network (CNN) has been immensely popular due to the introduction of deep learning. CNN is a supervised machine learning approach that can automatically learn robust, salient image category features from many labelled training images. Therefore, the neural network is used to automate image classification for websites. The studies [2,3,8-15] developed various CNN-powered vision models to detect and classify web GUI elements from visual designs.

DOM helps developers access and change different parts of the page, such as headings, paragraphs, images, and forms. Therefore, the DOM can be thought of as a way to organize and control all the elements that make up a web page. In the DOM, every element on a web page, like a heading or a paragraph, is treated as an object. Using objects, the DOM provides a consistent and organized way for developers to interact with web page elements. Therefore, classifying the web GUI elements of a webpage is an important step in automatically constructing a DOM hierarchy from a website mock-up design artifact.

2.2 Document Object Model (DOM) Hierarchical Construction

The related works show the methods of constructing a suitable DOM hierarchy for a website with proper grouping of web GUI elements in web GUI containers. Beltramelli [2] uses a Domain Specific Language (DSL) language model based on a Short-Term Long Memory (LSTM) network to encode classified web GUI elements and their semantic relationships into an intermediate DSL representation that represents the DOM hierarchy of the website. Alexander Robinson [6] introduced a recursive algorithm that recursively groups nested web GUI elements to build predefined web GUI containers. Finally, the hierarchical tree structure of the website is built by recursively grouping the classified web GUI containers. Nguyen et al. [7] introduced a method to detect and group mobile GUI elements into containers that generate hierarchical sub-trees. Then, the UI hierarchy for mobile applications is gradually constructed by merging those hierarchical sub-trees.

Yun et al. [12] proposed a method to construct the hierarchical structure of a UI sketch as an XML file utilizing the YOLO deep neural network, which extracted the structural and descriptive information of UI elements. Microsoft *Sketch2Code* [14] uses a layout construction algorithm to construct the DOM hierarchy of a website. Ferreira et al. [11] introduced a hierarchy reconstruction algorithm that iteratively integrates classified web GUI elements and containers to build the DOM hierarchy for a website. Similarly, to build the GUI skeleton from mobile design mock-ups, Chen C. et al. [15] utilized a Recurrent Neural Network (RNN) encoder and an RNN decoder with visual features of mobile GUI elements to produce the GUI skeleton from token sequence representation, and eventually converted to a UI hierarchy through Depth First Traversal (DFT) algorithm.

Although generating a hierarchical UI representation is an active research field as suggested by these breakthroughs, constructing the DOM from visual inputs is still a nearly unexplored research area. However, some research leverages machine learning to construct the DOM whereas some methods rely on heuristic algorithms. Our research presents another method of constructing the DOM from a screen capture of a website in collaboration with machine learning and heuristic algorithms.

3. APPROACH DESCRIPTION

The research study aims to develop a novel automatic DOM hierarchy construction approach that constructs DOM representations of websites from screen captures of real-world websites. Therefore, design science research methodology is followed to design and investigate a novel automatic DOM hierarchy construction approach as an artifact. Firstly, a series of image processing techniques (Canny's edge detection algorithm, morphological operations etc.) is applied to infer the boundaries of web GUI elements and to extract images of each GUI element from website screen captures. The output of this phase is a set of X-axis and Y-axis coordinates on a website screen capture. Similarly, based on the derived X-axis and Y-axis coordinates, a collection of web GUI element images cropped from the input website screen capture is generated and fed to the next phase to be classified into web domain-specific element categories.

After identifying the bounding boxes of web GUI elements in a website mock-up design artifact, the cropped images of those elements are classified into their respective web domain-specific categories. For that, a CNN image classifier is trained to classify the images of web GUI elements. However, no dataset contains labelled atomic web GUI elements. Therefore, a dataset is generated, including all structural information necessary to construct an artifact to detect and classify web GUI elements. Screen captures of current real-world websites are considered and categorized into three datasets used by the proposed CNN to train, validate and test the image classifier. As the initial prototype of this approach, pure HTML5 and CSS-supported websites are attempted, and 620 real-world website blocks are discovered from HTML website template providers on the internet as of December 2022.

Website blocks developed in pure HTML5 and CSS are manually filtered out, excluding HTML5 and CSS-supported website blocks developed in frameworks such as Bootstrap or PHP, website blocks with non-standard web GUI elements, such as gaming websites, tend to use non-standard interactive web GUI elements that can be complicated to categorize, and duplicate websites that belong to more than one category. Similarly, while online advertisements are a regular feature of most prominent websites, they are manually blocked when screen captures are captured to prevent detecting non-native elements such as flash, videos and frames. The website screen captures are captured using PhantomJs in MATLAB, a third-party screenshot capturer module, or with browser extensions such as *GoFullPage* for the Google Chrome browser and *Lightshot* screenshot tool for the Firefox browser. To maintain the consistency of height and width of all screen captures, the websites displaying in landscape orientation were reviewed, and their screen captures were verified at a resolution of 1280x1024.

To train the proposed CNN image classifier, a large set of images of GUI elements cropped from screen captures of real-world website blocks annotated with their appropriate web domain-specific types is required. For that, a web scraping tool is used to manually traverse the GUI of the target website in a manner similar to the Depth First Search (DFS) algorithm to capture the bounding rectangle coordinates, giving the precise locations, width, and height, and meta information of each GUI element, which are then stored in XML format. This information is utilized in MATLAB to extract individual sub-images of each web GUI element displayed on the website mock-up design artifacts saved in the dataset. Finally, the individual sub-images of each GUI element are manually labelled into their respective web domain-specific types with precious accuracy. This process produced a dataset containing screen captures of 620 distinct real-world website blocks with 33,120 sub-images of web GUI elements from 12 most prominent native classes.

The web GUI element images dataset is segmented into training, validation, and testing datasets on a scale for use with the suggested CNN image classifier. The segmentation methodology described in [16] research study is followed to segment web GUI element images into training, validation, and testing datasets in proportions of 75%, 15%, and 10%, respectively. The CNN architecture suggested is similar to the architecture of AlexNet CNN [17], which is one of the most commonly used convolutional neural networks for image classification. It has two fewer convolutional layers (3 instead of 5). The suggested CNN is developed in MATLAB utilizing three toolkits: Neural Network [18], Computer Vision [19], and Parallel Computing [20]. After training the CNN image classifier, new unseen images of web GUI elements can be fed to the classifier, resulting in a series of classification scores corresponding to each element class. With the proposed approach, the web GUI element class with the highest degree of confidence is assigned to be the appropriate label for a given target image. Section 4 evaluates the classification precision of the proposed CNN image classifier using the web GUI elements dataset described in this section.

The final stage in the approach is to build the DOM hierarchy of a website with the proper grouping of web GUI elements in web GUI containers. HTML5 offers six semantic layout components (header, nav, section, article, aside, and footer) as GUI containers to define different parts of a webpage. Therefore, an algorithm is designed to derive HTML5 semantic layout components by grouping corresponding GUI elements. The algorithm requires each GUI element's coordinates, width, and height in the input website image. The algorithm traverses the image along the X- and Y-axis directions to draw circumscribed rectangles, which are rectangles that touch at least one point of the GUI element with each of its slides, while all pixels of the GUI element contained within an internal area of the rectangle or its borders.

A circumscribed rectangle Recti is defined as a pair of coordinates that satisfy the following expression: top left corner (x_1^r, y_1^r) and bottom right corner (x_2^r, y_2^r). Assume that the coordinates of a pixel *p* are denoted by x^p and y^p, and the image of a GUI element is denoted by ele_i (top left corner (x_1, y_1) and bottom right corner (x_2, y_2)).

$$
\begin{aligned}
&(x_1^r, y_1^r), (x_2^r, y_2^r): \forall p \in ele_i\ (x_1^r \le x^p \le x_2^r \wedge y_1^r \le y^p \le x_2^r)\\
&\quad \wedge \exists p \in ele_i\ (x^p = x_1 \wedge y_1^r \le y^p \le y_2^r)\\
&\quad \wedge \exists p \in ele_i\ (x^p = x_2 \wedge y_1^r \le y^p \le y_2^r)\\
&\quad \wedge \exists p \in ele_i\ (y^p = y_1 \wedge x_1^r \le x^p \le x_2^r)\\
&\quad \wedge \exists p \in ele_i\ (y^p = y_2 \wedge x_1^r \le x^p \le x_2^r)
\end{aligned}
$$

Even if a web GUI element belongs to more than one circumscribed rectangle, every web GUI element should belong to exactly one circumscribed rectangle. Therefore, circumscribed rectangles can be assumed as web GUI containers containing any sequential web GUI elements. In constructing circumscribed rectangles, rectangles with intersection can be ignored because it is assumed that most web GUI containers in simple HTML5 and CSS-supported website blocks may not overlap. If a circumscribed rectangle satisfies the criteria of (I) not intersecting other circumscribed rectangles and (II) being structurally identical to one of the predefined HTML5 semantic layout components, it is extracted as a web GUI container with its associated elements. The algorithm is provided with a list of structural information of predefined HTML5 semantic layout components to ensure the derived containers are unique. The list of HTML5 semantic layout components comprises structural information of semantic layout components (x, y coordinates, width, and height) and their corresponding categories; the structural information is collected using a web scraping tool, and the corresponding categories are manually labelled. Finally, it generates a collection of HTML5 semantic layout components in the input website image, along with each layout component's structural information, category names, and GUI element composition. The pseudo-code of the algorithm can be explored in Algorithm 1 as follows. The result of the algorithm is depicted in Figure 1 and Figure 2.

Algorithm 1: Web GUI Container Derivation Algorithm

Input: *elements* - element$_1$(x_1, y_1, width$_1$, height$_1$) .... element$_n$ (x_n, y_n, width$_n$, height$_n$) and *HTML_containers* - a list of predefined HTML5 semantic layout components
Output: *containers* (a list of containers with associated elements)

function CONTAINER DERIVATION (*elements*)
    containers ← []
    temp.containers ← []
    for all $element_i \in elements$ **do**
        x^a ← min ($element_i^x$, $element_{i+1}^x$)
        y^a ← min ($element_i^y$, $element_{i+1}^y$)
        x^b ← max (($element_i^x + width_i$), ($element_{i+1}^x + width_{i+1}$))
        y^b ← max (($element_i^y + height_i$), ($element_{i+1}^y + height_{i+1}$))
        R ← rectangle (x^a, y^a) (x^b, y^b)
        if $element_{i+2} \cap R = \emptyset$ **then**
            $element_i^x$ ← rectangle (x^a, y^a) (x^b, y^b)
            continue
        end if
        temp. containers []← *R*
    end for
    for all $container_k \in temp.container$ **do**
        if $container_k \cap container_{k+1}$ = NULL
            if $container_k$ ← belongs to *HTML_containers* **then**
                containers ← $container_k$
            end if
        end if
    end for
    return *containers*
end function

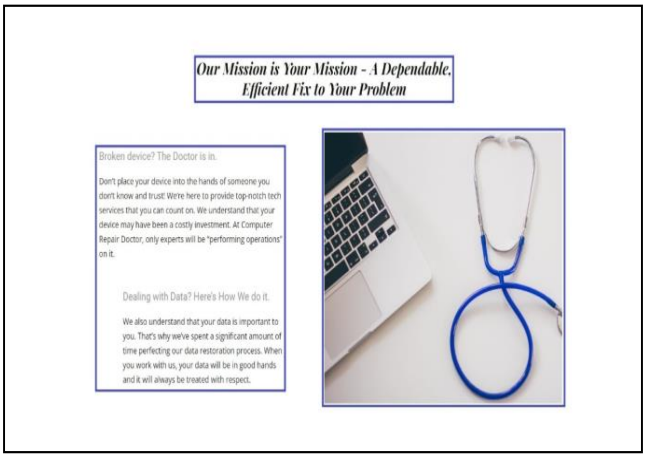


Figure 1: Examples of Subsets Identified as Containers on a

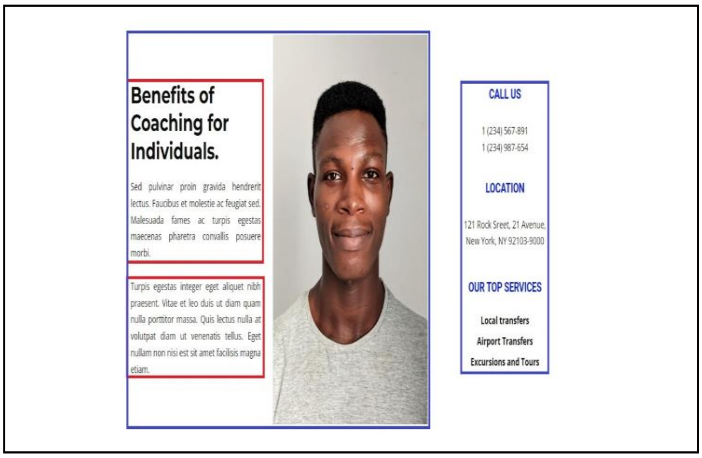


Figure 2: Examples of Subsets That Not Identified as

Accordingly, all subsets of web GUI elements with circumscribed rectangles in blue, as shown in Figure 1, are containers and correspond to predefined HTML5 semantic layout components, whereas subsets with circumscribed rectangles in red, shown in Figure 2, are not containers because of intersections between rectangles and do not correspond to any of the predefined HTML5 semantic layout components.

Once the GUI container components are derived, an algorithm is designed to construct the DOM hierarchy of a website with the proper arrangement of the derived web GUI containers and elements. For that, an iterative DOM hierarchy construction algorithm is designed to derive a realistic DOM hierarchy for a website gradually by utilizing derived web GUI containers as HTML5 semantic layout components, in which successive web GUI elements are grouped, and then constructing the DOM hierarchy with the proper layout of web GUI elements and containers.

According to the concept of child, parent, leaf, and internal nodes in graph theory, for instance, if N_1 and N_2 are web GUI elements and the coordinates of N_1 are greater than the coordinates of N_2 ($N_1 > N_2$), then N_2 is a child of parent N_1. Accordingly, nodes can be classified into two types based on their orientation property: *definite* and *indefinite*. A node is definite if its orientation can be uniquely defined as horizontal or vertical, whereas a node is indefinite if it has both horizontal and vertical orientation simultaneously or cannot be determined. The structure of a website screen capture is a tree in which each element corresponds to a definite node with a parent-child relationship, and each element must be connected to exactly one parent, except for the root node, which has none. The DOM structure of a website from a screen capture is constructed using that concept by suggesting an iterative DOM hierarchy construction algorithm where GUI containers are constructed as sub-trees, and the final tree is constructed by hierarchically arranging those sub-trees. The pseudo-code of the algorithm can be explored in Algorithm 2 as follows. The result of the algorithm is depicted in Figure 3.

Algorithm 2: DOM Hierarchy Construction Algorithm

Input: *containers* - a collection of derived HTML5 semantic layout containers for a website mock-up design artifact with their structural information, category name, and web GUI element composition

Output: |*containers*| == 1 and $containers_1$ - the root node of the tree containing all web GUI containers and web GUI elements for a website mock-up design artifact

function DOM CONSTRUCTION (*containers*)
  for all $containers_k \in containers$ **do**
    for all $element_i \in containers_k$ **do**
      orientation ← $element_i$.*orientation*
      if *orientation* is Horizontal **then**
        $element_j$ ← select the element to the right of $element_i$
      else if *orientation* is Vertical **then**
        $element_j$ ← select the element to the bottom of $element_{i+1}$
      if $element_j$ is not NULL **then**

if $element_i$ is *definite* **then**
make $element_j$ a child of $element_i$
else
newElement ← new *Element*
make $element_i$ a child of *newElement*
make $element_j$ a child of *newElement*
newElement.orientation ← *orientation*
$element_i$ ← *newElement*
$containers_k \leftarrow element \setminus element_j$
$i \leftarrow i + 1$
end if
end if
end for
$k \leftarrow k + 1$
end for
for all $containers_i \in containers$ **do**
orientation ← $containers_i^{root}$*.orientation*
if *orientation* is Horizontal **then**
$containers_i^{root}$ ← select the container to the right of $containers_1$
else
$containers_i^{root}$ ← select the container to the left of $containers_1$
end if
$i \leftarrow i + 1$
end for
return $containers_1$
end function

The algorithm intends to iteratively process each web GUI container to construct sub-trees with *parent-child* relationships of elements by merging each web GUI element into adjacent *definite* web GUI elements at each stage. The hierarchical arrangement of successive GUI elements within each GUI container is performed by evaluating their orientation according to the graph theory. Using MATLAB, hierarchy construction heuristics are developed to build sub-trees for web GUI elements to be properly placed in GUI containers. Initially, the algorithm produces a collection of sub-trees with related web GUI elements arranged hierarchically. The algorithm is further extended to construct a realistic DOM hierarchy for a website by iteratively arranging the constructed sub-trees like a tree. The algorithm terminates when exactly one component ($containers_1$) is constructed as the root representing a tree structure of web GUI containers and web GUI elements, each with a corresponding name (see Figure 3).

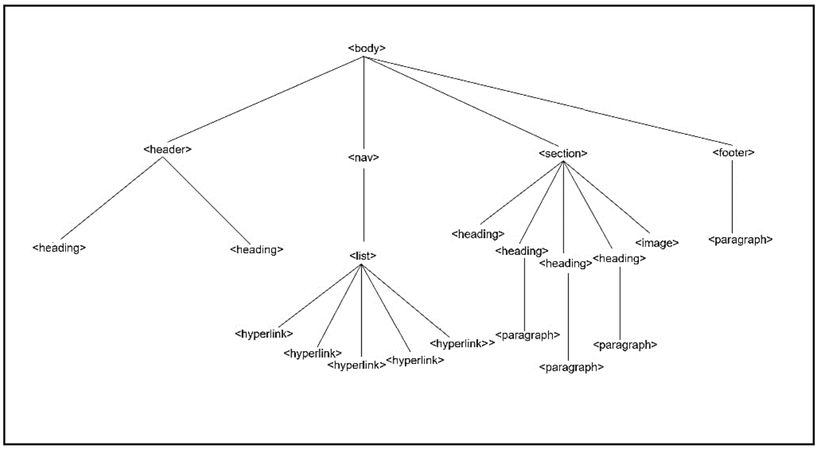


Figure 3: A Sample of DOM Hierarchy Constructed as a Tree Structure

4. RESEARCH EVALUATION

The suggested approach is evaluated on two validation strategies: (I) the accuracy of the CNN-driven classifier on web GUI element images classification and (II) the similarity of the DOM hierarchies constructed by the approach to real DOM hierarchies constructed by web developers. For the experiment of the approach, the study consists of (I) a set of 33,120 labelled images of web GUI elements extracted from 620 distinct website screen captures to assess the accuracy of CNN-driven web GUI element classifiers, (II) 64 additional website screen captures (excluded from the dataset used to train and test the CNN image classifier) to serve as website mock-up design artifacts, and (III) two latest approaches on automatic website generation: *Sketch2Code* [14] and *pix2code* [2].

4.1 CNN-Driven Image Classifier Effectiveness

Table 1 shows the datasets utilized, broken down by web GUI element class. The suggested CNN was trained on the training dataset and verified on the validation dataset. All the unseen web GUI element images in the testing and validation datasets were collected from the screen captures of real website blocks and were distinct from those in the training dataset. The effectiveness of the proposed CNN-driven image classifier is evaluated by measuring the average top-1 classification precision across all web GUI element classes in the test data set, as shown below.

$$\text{Precision} = \frac{\text{Number of predicted correct instances}}{\text{Total predicted instances}} = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positive}}$$

Here, *True Positives* are instances where the CNN image classifier correctly predicts positive web GUI element classes, whereas *False Positives* are instances where the classifier incorrectly predicts negative web GUI element classes. Section 5 describes the classification capabilities of the CNN-driven image classifier, which are represented in confusion matrix with precision across classes.

Table 1: Distribution of Labelled Web GUI Element Images on Training, Validation and Testing Datasets

Web GUI Element	***Total no of Elements After Data Cleaning***	***Training Data***	***Validation Data***	***Testing Data***
Heading	7,988	5,991	1,198	799
Paragraph	5,848	4,386	877	585
Image	3,696	2,772	554	370
Hyperlink	3,436	2,577	515	344
Dropdown Menu	256	192	38	26
Button	1,308	981	196	131
Google Map	138	104	21	13
Text Field	986	740	148	98
Text Area	378	284	57	37
Icon	3,242	2,432	486	324
Search Bar	216	162	32	22
Label	5,628	4,221	844	563
Total	***33,120***	***24,842***	***4,966***	***3,312***

4.2 DOM Hierarchy Construction

The similarity of DOM hierarchies constructed by the approach was measured by comparing to a ground truth set of DOM hierarchies and a set of DOM hierarchies constructed by two baseline website generation approaches: *Sketch2Code* [14] and *pix2code* [2]. For this part of the study, 64 website screen captures were selected from 620 distinct screen captures of real websites by randomly selecting four website blocks from each category to utilize as inputs to our approach, *Sketch2Code*, and *pix2code* approaches, from which DOM hierarchies were generated. To compare the DOM hierarchies constructed by each approach, the runtime DOM hierarchies extracted dynamically from the input websites for each approach using *Selenium WebDriver* are compared to the set of ground truth DOM hierarchies extracted from the original websites. The *Selenium WebDriver* representation of the website reflects the automatically generated GUI-related source code for each approach displayed at runtime on the device screen. It can be used to accurately compare the DOM hierarchy construction of web GUI elements and containers for each approach.

The runtime DOM-hierarchies of all generated prototype websites for each approach are compared to the original, ground truth runtime DOM-hierarchies (extracted from *Selenium WebDriver*) by deconstructing the DOM trees using pre-order traversal and using the Wagner-Fisher algorithm [21] implementation of *Levenshtein* edit distance to calculate the similarity between the DOM hierarchical representations of the runtime GUIs. The DOM hierarchies are deconstructed such that the element type and nested order of GUI elements are included in the hierarchy deconstruction. The pre-order traversal is implemented in this way to avoid small deviations in other attributes included in the *Selenium WebDriver* information, such as pixel values, given that the main goal of this evaluation is to measure hierarchical similarities. Three traditional edit operations supported by *Levenshtein* distance: insertion, deletion, and substitution are considered in the measurement of edit distance. To more completely measure the similarity of the DOM hierarchies of prototype websites to the ground truth DOM hierarchies, a weighting schema is introduced that represents a penalty value for each type of edit operation required to transform one GUI element of a DOM into another, wherein, in the default case, each edit operation carries an identical weight of 1/3.

5. RESULT DISCUSSION

5.1 Results: CNN-Driven Image Classifier Effectiveness

To evaluate the results produced by the CNN-driven image classifier, a confusion matrix is presented to illustrate the classification precision across the 12 most popular web GUI elements. Table 2 shows the confusion matrices generated by the CNN-driven image classifier. The X-axis of a confusion matrix represents the web GUI element class predicted by the classifier, and the Y-axis represents its actual web GUI element class. Similarly, the first column of the confusion matrices represents the total number of each web GUI element in the test dataset, while the numbers in the subsequent columns correspond to the percentage of each class on the Y-axis that were classified as web GUI elements on the X-axis. Thus, the diagonal of the matrix, highlighted in green (see Table 2), corresponds to the true classification (i.e., *True Positives*) of each web GUI element class, whereas *False Positives* are shown in the other cells.

Table 2: Confusion Matrix for CNN-driven Image Classifier

	Total	H	P	I	HL	DM	B	GM	TF	TA	IC	SB	L
H	799	86%	2%	0%	5%	2%	1%	0%	0%	0%	0%	0%	4%
P	585	15%	84%	0%	1%	0%	0%	0%	0%	0%	0%	0%	0%
I	370	0%	0%	100%	0%	0%	0%	0%	0%	0%	0%	0%	0%
HL	344	4%	1%	0%	78%	14%	1%	0%	0%	0%	0%	0%	2%
DM	26	5%	0%	0%	19%	72%	3%	0%	0%	0%	0%	0%	1%
B	131	0%	0%	0%	7%	15%	65%	0%	0%	0%	0%	0%	13%
GM	13	0%	0%	0%	0%	0%	0%	100%	0%	0%	0%	0%	0%
TF	98	3%	0%	0%	1%	0%	0%	0%	93%	3%	0%	0%	0%
TA	37	4%	6%	0%	0%	0%	0%	0%	7%	83%	0%	0%	0%
IC	324	0%	0%	0%	0%	0%	0%	0%	0%	0%	100%	0%	0%
SB	22	8%	2%	0%	6%	0%	4%	0%	2%	0%	0%	78%	0%
L	563	9%	3%	0%	5%	0%	2%	0%	0%	0%	0%	0%	81%

Column Heading Abbreviations for Web GUI Elements: Heading (H), Paragraph (P), Image (I), Hyperlink (HL), Dropdown Menu (DM), Button (B), Google Map (GM), Text Field (TF), Text Area (TA), Icon (IC), Search Bar (SB) and Label (L).

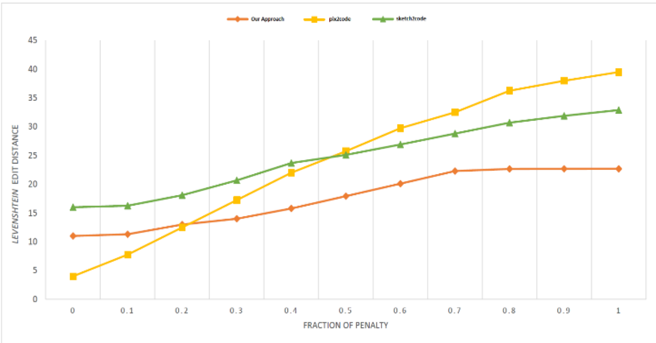
The CNN-driven image classifier achieves 89.8% of the top-1 precision on the prediction of web GUI elements, which is the overall probability of true classifications. Therefore, it is experimentally stated that the CNN-driven image classifier employed in the proposed approach performs much better.

5.2 Results: DOM Hierarchy Construction

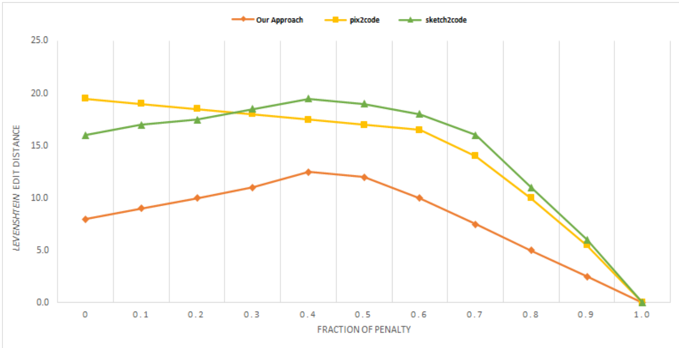
An important step in the DOM hierarchy construction process is the automatic generation of a DOM hierarchy, enabling a well-structured layout and the proper visualization of web GUI elements into HTML5 web GUI containers. The evaluation of the approach's DOM hierarchy construction compares against the two baseline website generation approaches, *Sketch2Code* and *pix2code*, by decomposing the runtime DOM-hierarchies into trees and measuring the Levenshtein edit distance between the generated trees and target trees as described in Section 4.2. By varying the penalty values prescribed to each edit operation, a more comprehensive understanding of the similarity of the generated DOM hierarchies can be gained by observing, for instance, whether certain DOM hierarchies were more or less shallow than the real websites, by examining the performance of insertion and deletion edits.

Figure 4 compares the DOM hierarchies generated by our approach, *Sketch2Code*, and *pix2code* approaches based on tree edit distance (I – III), which measures the minimum cost of editing operations when transforming one tree into another. Each graph illustrates results for a different edit operation, and the proposed approach, *Sketch2Code*, and *pix2code* approaches are represented by lines in three different colours with the edit distance shown on the Y-axis and the penalty prescribed to the edit operation on the X-axis.

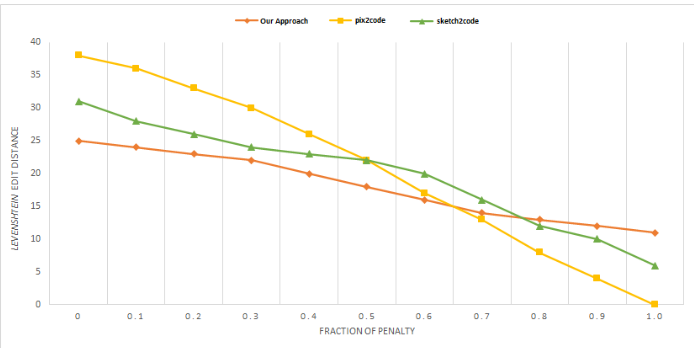
The graphs show that the *Sketch2Code* and *pix2code* approaches trend to produce relatively shallow DOM hierarchies.



(I) Insertion Edits



(II) Substitution Edits



(III) Deletion Edits

Figure 4: Similarity of DOM Hierarchies Constructed by WebDraw, pix2code and Sketch2code based on Levenshtein Edit Distances

When the penalty for insertion operation is increased, our approach outperforms both *Sketch2Code* and *pix2code*. This is because our approach requires fewer insertion operations into the DOM hierarchies to match the target DOM hierarchies precisely. However, *Sketch2Code* and *pix2code* require adding more inner nodes to the DOM trees because they produce very shallow DOM hierarchies. The evaluation shows that the shallow DOM hierarchies produced by the *pix2code* and *Sketch2Code* approaches match the target DOM hierarchies and our approach's DOM hierarchies. When rendering a website in a web browser, a shallow DOM hierarchy is preferred, and our approach builds DOM hierarchies that are significantly similar to target DOM hierarchies and tend to be slower than *Sketch2Code* and *pix2code*.

6. FUTURE DIRECTIONS AND CONCLUSIONS

Although the proposed approach is an effective tool for constructing DOM hierarchies, it has some practical limitations, some of which present future research opportunities in automatic DOM hierarchy construction. When websites are screen-captured from desktop or laptop computers, the iterative DOM hierarchy construction algorithm is currently tied to the website's appearance on a particular screen size. Although this algorithm can be generalized to work regardless of screen size using display density-independent pixel (dp) values, this is a task for the future.

The current CNN-driven web GUI element classifier can classify incoming images into one of the top 12 web GUI elements. Thus, it currently does not support unusual web GUI element types. Future research studies could examine CNN architectures with greater classification capacity (for instance, CNN architectures in deep learning to classify more web GUI elements with web GUI containers), or researchers could even examine newly emerging architectures such as hierarchical CNNs [22]. The DOM hierarchy construction approach currently operates in two steps to detect and classify web GUI elements from a website mock-up design artifact; however, future research studies will investigate the potential of object detection classifiers in CNN [23,24] that can perform these object detection and classification steps simultaneously.

The ever-growing field of machine learning has led researchers to a significant step forward: improving UI designs to provide a better functional user experience at a lower cost. With the discovery of the approach, researchers can explore object detection-oriented CNN classifiers in the future to provide better support in the web GUI elements detection phase and object classification. Similarly, the approach can be deployed as an automatic website prototype generation tool that converts the constructed DOM hierarchies into functional HTML/CSS code.

The novel automatic DOM hierarchy construction approach presented in this paper will benefit web UI/UX developers by allowing them to speed up the iterative web UI prototyping process in their workflow by enabling the generation of automatic DOM for a website format of their choice. Therefore, the inventive approach meets the need to introduce a novel approach to the automatic DOM hierarchy construction strategy in the automatic web UI prototyping concept to reduce the burden and waste of time in the iterative web UI prototyping process.

ACKNOWLEDGEMENT

This research received no specific grant from funding agencies in the public, commercial, or not-for-profit sectors.

REFERENCES

[1] Kaluarachchi, T., & Wickramasinghe, M. (2023). *A systematic literature review on automatic website generation*. Journal of Computer Languages, Volume 75, 2023, 101202, ISSN 2590-1184, https://doi.org/10.1016/j.cola.2023.101202

[2] Beltramelli, T. (2018, June). *pix2code: Generating code from a graphical user interface screenshot*. In Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems (pp. 1-6).

[3] Suleri, S., Sermuga Pandian, V. P., Shishkovets, S., & Jarke, M. (2019, May). *Eve: A sketch-based software prototyping workbench*. In Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems (pp. 1-6).

[4] Moran, K., Bernal-Cárdenas, C., Curcio, M., Bonett, R., & Poshyvanyk, D. (2018). *Machine learning-based prototyping of graphical user interfaces for mobile apps*. IEEE Transactions on Software Engineering, 46(2), 196-221.

[5] Ozkaya, *Are DevOps and automation our next silver bullet?* IEEE Softw. 36 (4) (2019) 3–95.

[6] Robinson, A. (2019). *Sketch2code: Generating a website from a paper mock-up*. arXiv preprint arXiv:1905.13750.

[7] T. A. Nguyen and C. Csallner, *Reverse Engineering Mobile Application User Interfaces with REMAUI (T)*. 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE), Lincoln, NE, USA, 2015, pp. 248-259, DOI: 10.1109/ASE.2015.32.

[8] Hassan, S., Arya, M., Bhardwaj, U., & Kole, S. (2018). *Extraction and Classification of User Interface Components from an Image.* International Journal of Pure and Applied Mathematics, 118(24), 1-16.

[9] Aşıroğlu, B., Mete, B. R., Yıldız, E., Nalçakan, Y., Sezen, A., Dağtekin, M., & Ensari, T. (2019, April). *Automatic HTML code generation from mock-up images using machine learning techniques*. In 2019 Scientific Meeting on Electrical-Electronics & Biomedical Engineering and Computer Science (EBBT) (pp. 1-4). IEEE.

[10] Kim, B., Park, S., Won, T., Heo, J., & Kim, B. (2018, October). *Deep-learning based web UI automatic programming*. In Proceedings of the 2018 Conference on Research in Adaptive and Convergent Systems (pp. 64-65).

[11] J. Ferreira, A. Restivo, H.S. Ferreira, *Automatically Generating Websites from Hand-Drawn Mock-Ups*, In Proceedings of the 16th International Joint Conference on Computer Vision, Imaging and Computer Graphics Theory and Applications, 2021.

[12] Yun, Y. S., Jung, J., Eun, S., So, S. S., & Heo, J. (2019). *Detection of GUI elements on sketch images using object detector based on deep neural networks.* In Proceedings of the Sixth International Conference on Green and Human Information Technology: ICGHIT 2018 (pp. 86-90). Springer Singapore.

[13] Kaluarachchi, T. T., Kumara, H. M. M. S. D., & Wickramasinghe, M. I. E. (2020). *Turning GUI into HTML source code with Image processing and Deep learning*. PGIS International Research Congress (RESCON) 2020, Faculty of Science, University of Peradeniya, Sri Lanka.

[14] Sketch2Code - Microsoft AI labs, 2019, https://sketch2code.azurewebsites.net/

[15] Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, Yang Liu, *From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI Implementation*, in: Proceedings of the 40th International Conference on Software Engineering, ICSE '18, ACM, New York, NY, USA, 2018, pp. 665–676.

[16] M. D. Zeiler and R. Fergus. *Visualizing and Understanding Convolutional Networks*. Cham: Springer International Publishing, 2014, pp. 818–833. [Online]. Available: https://doi.org/10.1007/978-3-319-10590-1_53

[17] A. Krizhevsky, I. Sutskever, and G. E. Hinton. *Imagenet Classification with Deep Convolutional Neural Networks*. In Advances in Neural Information Processing Systems 25, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[18] MATLAB Neural Network Toolbox: https://www.mathworks.com/products/neural-network.html

[19] MATLAB Tesseract OCR Library: https://www.mathworks.com/products/computer-vision.html

[20] MATLAB Parallel Computing Toolbox: https://www.mathworks.com/products/parallel-computing.html

[21] Wagner-Fischer algorithm: https://en.wikipedia.org/wiki/Wagner-Fischer_algorithm

[22] R. Girshick, J. Donahue, T. Darrell, and J. Malik, 2014. *Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation*. In Proceedings of the 2014 IEEE Conference on Computer Vision and Pattern Recognition, ser. CVPR '14. Washington, DC, USA: IEEE Computer Society, pp. 580–587.

[23] S. Ren, K. He, R. Girshick, and J. Sun, 2015. Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. In Proceedings of the 28th International Conference on Neural Information Processing Systems, ser. NIPS'15. Cambridge, MA, USA: MIT Press, pp. 91–99. [Online]. Available: http://dl.acm.org/citation.cfm?id=2969239.2969250

[24] Gu, Jiuxiang, Zhenhua Wang, Jason Kuen, Lianyang Ma, Amir Shahroudy, Bing Shuai, Ting Liu et al., 2018. *Recent Advances in Convolutional Neural Networks*. Pattern recognition, 77, 354-377.

BIOGRAPHIES

Thisaranie Kaluarachchi received a B.Sc. degree in Computer Science from the Faculty of Science at the University of Peradeniya in Kandy, Sri Lanka. She is pursuing her PhD in Computing at the University of Colombo School of Computing in Colombo, Sri Lanka. Her research interests include Artificial Intelligence, Machine Learning, Deep Learning, and Computer Vision.