International Research Journal of Engineering and Technology (IRJET) e-ISSN: 2395-0056
Volume: 11 Issue: 12 | Dec 2024 www.irjet.net p-ISSN: 2395-0072

A CNN Classifier for HTML Code Generation from GUIs

Thisaranie Kaluarachchi

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

---

Abstract - *Internet usage has increased exceedingly and rapidly in the past decades. The World Wide Web (WWW) has emerged as the most important public communication portal for individuals, businesses and organizations. In recent years, web development has relied on website templates and Content Management Systems, which anyone can use to create a website by plugging in their own text and images. Similarly, web designers create a design mock-up for a web page and give it to a developer to implement in code. This process is challenging and time-consuming since the design and implementation are carried out by separate teams, which is costly. The overall outcome of such a design depends entirely on the web designer's design skills, and there are differences between two source codes that have been implemented in the same UI by two different developers. What if there was a mechanism or system that could detect HTML elements in GUI images and generate source code automatically? If such a system or tool can generate source code automatically, web developers can focus on functionalities rather than wasting time on front-end development. Consequently, this paper presents an approach that automatically converts the GUI design of a website into HTML code using image processing and deep learning. This study employed an experimental approach to realize two scenarios that operates as image processing and deep learning modules. The tag tree produced by the approach was compared to the original HTML tag tree, and the HTML code produced for a specific website was compared to the website's original HTML source code.*

Key Words: deep learning, image processing, HTML code generation, web designing, automatic website generation

1. INTRODUCTION

Front-end UI development based on GUI design is the primary responsibility of developers in website development. Generally, this process takes longer than achieving system functionality and logic. Therefore, it preserves developers from focusing on implementing key functionality. Also, the methods of implementing such a UI are determined by the developer's experience and skills. As a result, even when two different developers implement the same UI, there are differences in the source code. What if there was a mechanism or system that could detect HTML elements in GUI images and generate source code automatically? If such a system or tool can generate source code automatically, web developers can focus on functionalities rather than wasting time on front-end development. Furthermore, the system can follow the standards and rules. Therefore, the output of the system will produce standards-compliant source code, which can provide additional benefits to the website that implements it.

User interface (UI) design takes into account the needs of end users, ensuring that the system is packed with elements that make it easy for users to access and understand its features. In addition, today's primary focus is on GUI design using various images, effects and animations. GUI design and implementation, however, are challenging and time-consuming [1]. Front end development for websites or web apps is more complex than GUI implementation. It entails working with a variety of technologies and languages, including HTML, JavaScript, PHP, ASP.NET, MySQL, and AJAX. There may be instances where developers become trapped for a few hours or days [2]. When it comes to front-end layout implementations, web elements are classified into two categories: fix GUI elements (buttons, text inputs, paragraphs, etc.) and dynamic elements (dropdowns, drawer menus, etc. In this study, we concentrated solely on converting fixed GUI elements into source code as the purpose of this research is to develop a solution for converting GUI into source code using image processing and machine learning.

End-users of a system interact with it through the user interface. The user interface is the primary gateway through which users communicate with the system. It is vital and front-end development receives more attention from the Software Development Life Cycle (SDLC). Front-end developers must work hard to build GUI elements to accomplish the unique design provided by graphic designers. Developers spend a lot of time choosing the appropriate HTML tags, planning the proper sequence and structure, then coding. Furthermore, developers may have to repeat the same sort of code lines numerous times to construct a single page of GUI which is a common issue in markup languages. Therefore, it is time-consuming and depend on the developer's experience and skills.

The primary aim of the research is to design, develop, and evaluate an approach for converting GUI design graphics into HTML code. The research focused on techniques from image processing and deep learning to design and implement the approach. The study makes the following contributions:

- Constructing a mechanism to extract HTML elements from a GUI design.
- Constructing a mechanism to classify the HTML tags.
- Constructing a CNN to generate source from HTML tag hierarchy tree.

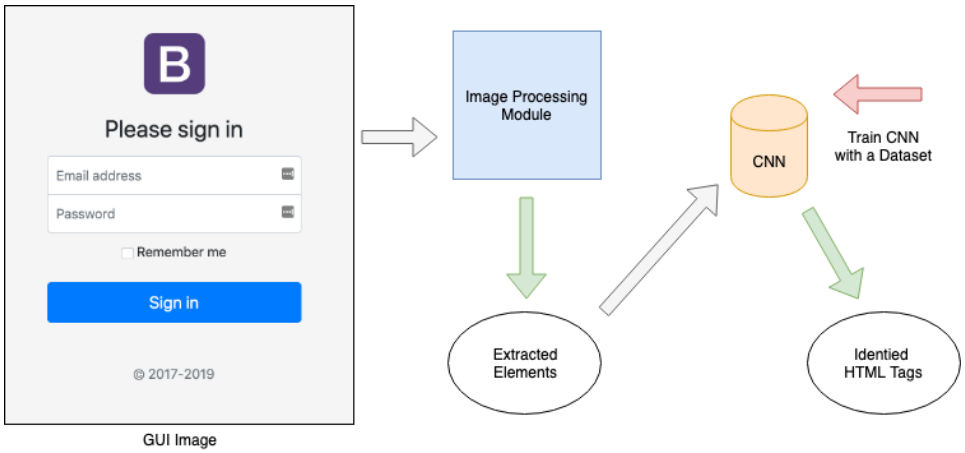


Figure 1: Conceptual Design of The Proposed System

Figure 1 illustrates the overview of the proposed approach for HTML Code Generation from GUIs, which is referred to throughout the description. A detailed description of the approach is described in Section 3. The remainder of this paper is organized as follows: Section 2 conducts a literature review to comprehend the concepts related to the problem domain and scope. Furthermore, similar studies involving the conversion of a GUI into source code are discussed. Section 3 describes the proposed research methodology and the theoretical basic of the design and concept evaluation to arrive at a proper answer. Section 4 thoroughly covers the design concept and the image processing and deep learning techniques used in the solution. 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 HTML Code Generation approach.

2. LITERATURE REVIEW

The scope of study is divided into four sub-scopes namely graphical user interfaces, code generation, computer vision and deep learning. Accordingly, the literature review section analyses the effectiveness of converting GUI to source code and critically evaluates comparative work in the problem domain. Furthermore, it critically examines the various strategies, technologies, algorithms, methodologies, and tools available that can be used to effectively and efficiently produce source code from GUIs.

2.1 Graphical User Interface (GUI)

The user interface connects end users to the system or program and allows them to interact with the system or software. Myers [1] shows that small details of the system need to be demonstrated and significant to have an impact on the overall system for the end users. Furthermore, he has outlined many approaches to determine how well UI design systems impress end users. UI design plays an important role in representing the software and all its purposes. If the system UI is not designed with sufficient amount of information, the final software system will not be able to interact effectively with the users.

A GUI is an expanded version of the UI described in the previous paragraph. End users can interact with a system or software using visual prompts instead of a text-only UI [3]. GUI introduced a technique for graphically interacting with software systems. It replaced the majority of Command Line Interfaces (CLIs) [4]. GUIs combine software and hardware to create an interactive platform for end users to collect, produce, and display data and information. As a result, GUI development became a key part of the software development process.

When system development comes to the use of a web browser, it is usually referred to as a website. The GUI of the web interface allows users to interact with the respective website. The website's GUI is mainly used to represent data as well as input to the web system.

2.1.1 User Interface Elements

The parts of a GUI are user interface elements. These are components that represent data in the system in a structured way. These components assist users to interact with the software system without in-depth experience or knowledge of technology manipulation and computer skills. Most of the visual agents that GUIs use to capture data, information, and user input are static components, but some of them are dynamic, meaning they change periodically depending on conditions and interact directly with end users to make them feel the real-time communication with computer systems [4]. Generally, GUI elements include the following,

- **Informational Components**: tooltips, icons, progress bar, message boxes, modal windows, notifications
- **Input Controls**: text fields, checkboxes, radio buttons, list boxes, buttons, toggles, date field
- **Navigational Components**: breadcrumb, slider, search field, pagination, slider, tags, icons
- **Containers**: accordion

Most of these web GUI elements can be classified under two main categories, static UI elements and dynamic UI elements. Faure et al. [5] have briefly described the difference between static element arrangement and dynamic element arrangement in a system UI. Static element arrangement refers to elements that are fixed components of a UI and these arrangement elements are mainly used to represent system data to users and receive user actions as inputs. Note Elements that are dynamically generated with animation, timing, or user interaction can be included in the static element category. Dynamic elements are those with real-time information that changes from time to time in the system due to time-to-time changes or user interaction.

2.2 Markup Language

Structured method of text manipulation in a computer calls sign language. These methods make text easily recognizable and human readable. Markup language is not a programming language, but a method of creating a structure of text or a document for presentation on an electronic device. It can be used by an operating system, application, or program to present data as text, images, and other visual components. Markup languages [6] are classified into three categories as follows.

- **Presentation markup:** Traditional word processing systems used with WYSIWYG. It is hidden from human users. It includes horizontal and vertical spacing such as page breaks, double line spacing, and indentation. It is generated by a mechanical or electronic typewriter.
- **Procedural Markup:** It integrates with text and provides text processing instructions to programs. Such text is handled visually by the author. Procedural markup systems include programming constructs where macros or subroutines are defined and called by name.
- **Descriptive Markup:** This category is concerned with the logical structure of a document rather than its appearance. Descriptive symbols are designed to be easy for people and computers to read and understand. User-defined styles in word processors can be used for descriptive markup.

2.2.1 HTML

The syntax associated with HTML today is www. It is a network of information resources called the World Wide Web. Dave Raggett et al. [7] have proven that there are three main components involved in this worldwide network as follows, and HTML comes under the Hypertext mechanism.

- Consistent naming convention for accessing web resources directly (URIs).
- Protocols (HTTP, HTTPS) to access resources over the Internet.
- Hypertext method to simply navigate through resources and represent data (HTML).

HTML is the markup language most used by developers to implement web GUIs. HTML explains and defines how text, images and multimedia are displayed in web browsers [8]. There are two types of tags in HTML:

- **Paired Tags:** A pair tag consists of two tags, the first one is called the opening tag and the second tag is called the closing tag. These tags contain the text that should apply the effect of that tag.

- **Unpaired Tags:** An unpaired tag is a single tag that does not require a companion tag. These tags are written as < > or < /> to work with each other. Which style to choose is the developer's choice.

Most page structuring tags and formatting tags are paired tags. Therefore, each of them has both opening and closing tags. When implementing a GUI with HTML tags, it can be represented as a hierarchical tree as shown in Figure 2.

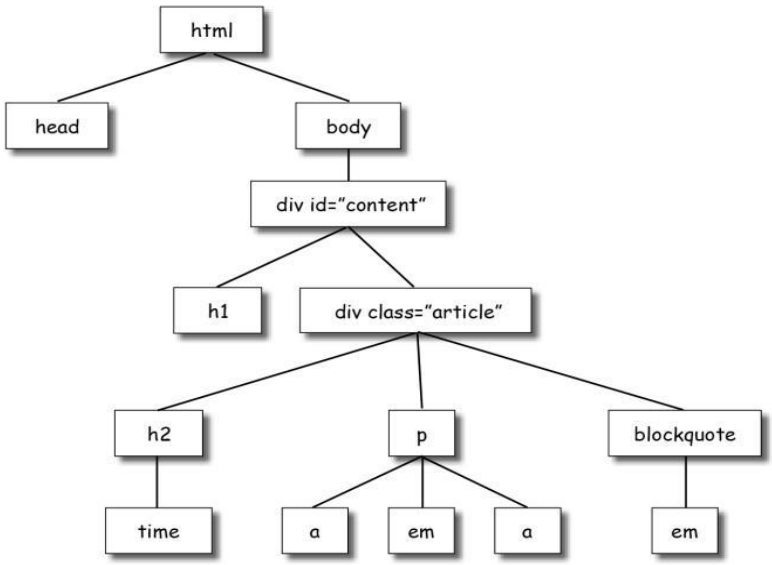


Figure 2: Hierarchical tree structure of HTML tags

2.2.2 Scripts

Developers generally use client-side scripting by embedding it with HTML in web application development to reduce the time of data processing and data rendering of a website or web page. This script runs on the client-side computer. Therefore, the system does not block the user from waiting for the server-side process. Many animations, simple logic, and even local storage values are handled with these server scripting methods. Similarly, scripts are used to create GUI elements in web design. Furthermore, Dave Raggett et al. [7] have explained how to come up with this script form to process inputs while handling user input.

2.2.3 Style sheets

Dave Raggett et al. [7] have shown how HTML code handles the responsibility of rendering web views by simplifying HTML with style sheets in web development. Stylesheets allow users to control alignment, font, color, and layout settings. Developers can use them in HTML code or as a separate file that can be referenced externally. The method of associating the style sheet with the web page is independent of the styling language they use.

2.2 Code Generation

Generally, in computer science, SDLC is the structure imposed on software development by a development method [9]. Synonyms include software development and software process. Similarly, in the UI field, UI Development Life Cycle (UIDLC) consists of the development path defined by a UI development method for developing UI [10]. The implementation phase creates an actual website from site design. As a first step, the elements and relationships highlighted in the design are mapped onto the constructs provided by the chosen implementation technique [11]. Turn the GUI design created by a graphic designer into a source code that comes under front-end UI development. GUI implementation is one of the main tasks of developers in the website development process.

2.2.1 Human coding and Front-End Development

Traditionally GUI development and implementation are a human task, and it takes time to fully convert a GUI design into source code. Often, this process takes more time than realizing system functionality and system logic. Moreover, this process

takes more time and prevents developers from focusing on implementing key functionality [12]. The time spent may vary on the same implementation, depending on the skill level of the developer. In the absence of a disciplined approach to Web-based system development, that Web-based applications are not delivering desired performance and quality, and that development process becomes increasingly complex and challenging to manage and refine and also expensive and grossly behind schedule [13].

2.2.2 User Interface Tools

Generally, developers use user interface implementation tools for GUI implementations. Developers get more benefits by using these tools. There are two main categories of UI tools as described below [14].

First, the quality of the resulting user interfaces can increase because designs can be rapidly prototyped and implemented even before application code is written. Because of that high iteration of rapid design, it's an important part of achieving user interface quality. The code of the UI is automatically created with high-level specifications, so the reliability of the UI will increase. When creating UIs using UI generator tools, most of the end products are the same, even if the applications are different. Those tools are easy to use, so non-programmers can work on UI implementation. People such as graphic artists, usability studies specialists, and cognitive psychologists can also be involved in this process.

Second, because interface specifications can be represented, validated, and evaluated more efficiently, UI code can be easier and more cost-effective to create and maintain. Since UI tools play a key role in implementing the UI, writing code will be less work. Also, there can be better modularization due to separate implementation between UI component and application implementation. Generating web UIs uses different tools than building typical desktop or mobile software GUIs. Simple web pages are created with consistent text, graphics and links. These simple components can be created by writing HTML codes directly. But when UI generation tools are used to design web systems, they can create pages that contain text, buttons, input fields, etc. Also scripting languages should be used in HTML to implement dynamic pages.

2.2.3 Transforming Graphical Designs into Code

Many of the techniques used in converting graphical images into codes can be categorized into machine learning streams. More recently, there have been several attempts to use machine learning techniques such as deep learning neural networks. Among these efforts, some focused on structured markup languages [16].

Ali Davody et al. [17] have introduced a Reinforcement Learning (RL) method that generates HTML source codes when the image is provided on the web page. RL framework agent is typically trained to maximize expected evaluation metrics instead of maximizing conditional probability tokens. The agent generates code that renders a web view that better matches it. The model they proposed is an agent that can generate HTML source code that takes a website image as input and then renders it back in a browser. They use an RL approach, where the agent model receives a token at each timestamp and creates a model that can simply generate a DSL source and is trained to capture DSL tokens. They also explained a method to analyze the generated website screen with the target image and its results. For their method, they only use the web browser to calculate the reward, making the whole process independent. They also conducted some experiments on a synthetic database built with DSL support. It generated simple HTML web pages for a few tokens [17].

Alexander L. et al. [18] have developed another technique based on deep learning to solve the frontend UI generation problem. They used predictions from neural networks to augment search methods including an SMT-based solver and enumeration searches. From further observations, they have shown that their method leads to non-robust baselines and an order of magnitude faster than the RNN method. They have proposed two main ideas to generate computer programs using mathematical predictions to augment common search methods. The first idea is how the system learns to produce programs that use a collection of program initialization difficulties to study policies that terminate on difficulty. The latter explains how neural network architecture can be implemented by prioritizing search-based methods over replacement methods. For solving Inductive Program Synthesis (IPS), they have shown that machine learning can provide significant value. Learning from the association of input and output samples across different interpreters can enable source code generation. They also used DSL to reduce the complexity of the programming language and it was able to reduce the search space area [19].

Moreover, Ling et al. [20] have recently studied inputs through a mixture of natural language and structured program specifications and then explained the organization of the program. The most common factor was the use of DSLs designed to target specific domain. They differ from full-fledged computer languages and are more restrictive. It reduces the complexity of the language used in DSLs. Generating source code from visual inputs is an unexplored area of research.

Another closely related work using reverse engineering techniques was implemented by Nguyen et al. [21], focusing on Android mobile application developments. It was able to regenerate the source code of the UI by taking the Android app screenshot as input. Their method is completely dependent on engineering background and requires good domain knowledge.

pix2code [12] is the first attempt to solve the problem of creating web page UI codes through visuals as input by taking machine learning support to find potential variables other than implementing complex methods. It focusses on another CNN-based approach, which generates tokens for the final stage from a single GUI screenshot input. Furthermore, it compares the process of generating source code from an image or screenshot to describing an image in English and writing it. Both of these processes require the creation of variables of tokens by analyzing the image pixel by pixel. It tried on a small dataset by scraping scenes and training a few parameters [22,23].

2.3 Computer Vision

All of the technologies and methodologies mentioned above rely on computer vision-related recognition and visual classification. This subsection discusses the image processing techniques that will be employed in the following steps. Other researchers have conducted several trials and attempts in image processing and edge detection, with the LoG approach providing superior answers [24]. Image edge detection with the Gaussian Laplace operator can mimic the visual properties of human eyes. In practice, noise has a significant impact on an image since the scale factor is incapable of self-adaptive modification [25].

2.4 Deep Learning

Deep learning principles must be considered with image processing techniques. In terms of image identification and classification, CNN ranks highly among neural networks. Currently, there are various machine learning approaches to computer vision problems, but CNN is the method of choice and is used in a wide range of applications. This architecture helps the network to learn sharp and rich potential representations through the training images. [26][27].

3. METHODOLOGY

To attain the end goals, the study used an experimental approach. The process of converting GUI into source code is divided into two modules. The first module is the image processing module, which extracts HTML elements from the GUI image and determines the hierarchy of elements. The second module is for recognizing HTML tags derived from a GUI image.

3.1 Image processing module

The MATLAB image processing toolkit was used to construct the image processing module, which was designed to extract HTML elements from a GUI image. The extractor module was designed using image processing techniques including LoG, dilate, and erode. Experiments were conducted by adjusting the variable settings for each stage and comparing the processed output images to the input GUI images. Furthermore, experiments were conducted to increase the number of extracted HTML elements from the GUI image and compared it to the number of HTML element tags in the original, which provided the input GUI for the experiment. Figure 3 summarizes the concept and experiment of the image processing module. The Notepad++ utility is used to compare the counts of two code segments.

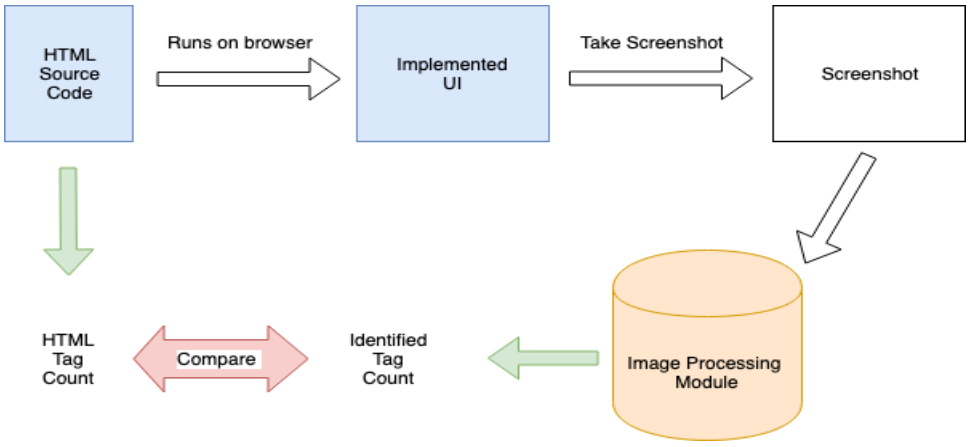


Figure 3: Overview of the Image Processing Module

3.2 Deep learning module

TensorFlow and Python technologies were used to create the deep learning module. PyCharm IDE was used to implement the Python code for CNN development on a local Mac PC running Linux. Recently, the module was hosted on an AWS cloud instance to improve prediction accuracy. Figure 4 describes the high-level design of the deep learning module.

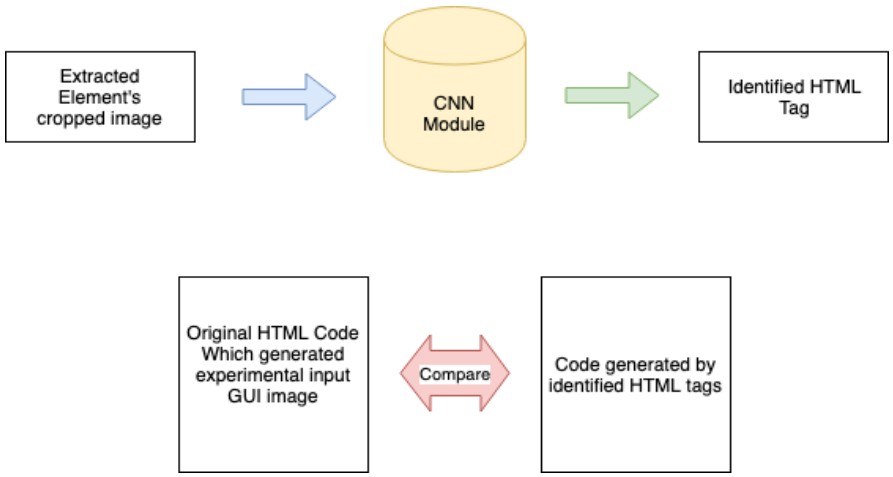


Figure 4: Deep Learning Module Architecture

4. EXPERIMENTAL APPROACH

The proposed solution is organized into three main components. Figure 5 depicts the flow of image processing, markup tag structure, and HTML source code production at a high level. The proposed system's final product will comprise an HTML styling unit, which may include CSS or Sass. However, this scope only included HTML tag production.

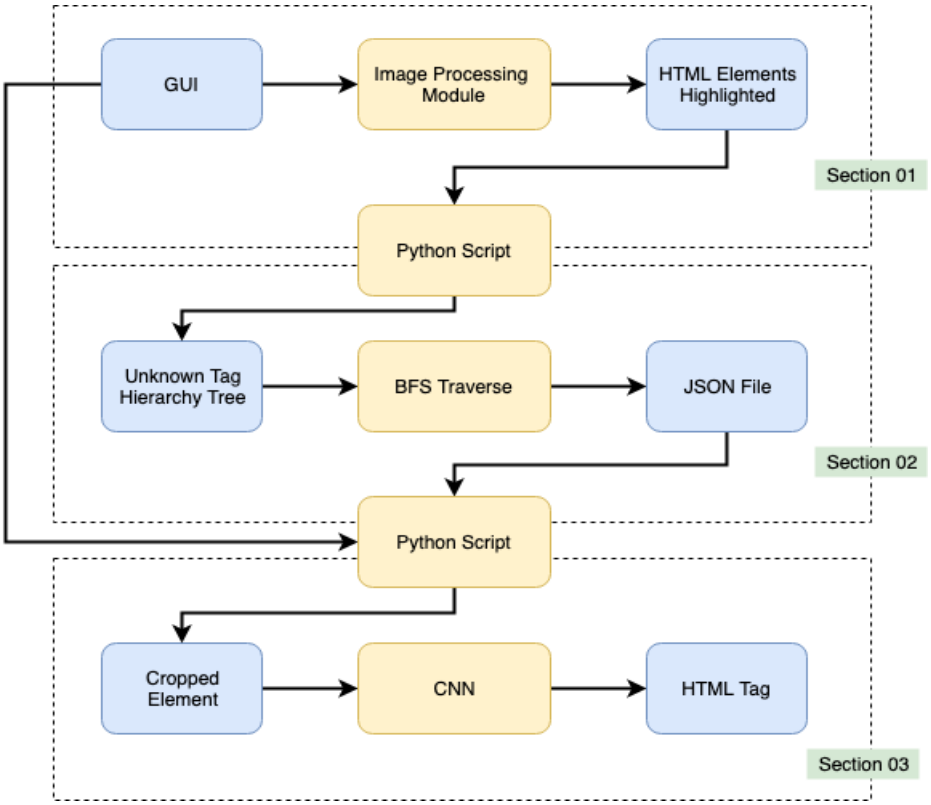


Figure 5: High-Level Architecture of the Approach

4.1 GUI Image Processing and Elements Detection

The first module of the proposed solution can be named as image processing unit. When a GUI image is given as input, this unit processes the image in several steps and returns an image with the detected HTML elements. Figure 6 illustrates the unit input and predicted processing output. To extract HTML elements from the input GUI image, the edge detection algorithm of MATLAB [28] was used. Furthermore, the unit of this proposed solution can be described as four image processing steps leading to the output of this image processing subunit. Each step is described below.

Figure 6: Sample Input GUI and Expected Output

4.1.1 Laplacian of Gaussian filter (LoG)

The input GUI image was subjected to a LoG filter in the initial stage of the image processing subunit. It uses *fspecial* method with parsing *'log'* argument to create a 2-D filter on the input image. This derivative filter is used to locate edges in images by identifying areas of fast change [29].

4.1.2 Convert Image into Binary Image

The second stage of the image processing unit is to generate a binary image. The previous step's output is used as input for this operation. *Im2bw* was used to convert the indexed data into binary images. It first converts the input image to grayscale format and then converts it to binary image. The image generated by this step contains only black pixels. To complete this step, use MATLAB's *im2bw* function [30] to parse the *'level'* input. The level can be specified in the range [0,1], and the value used is about 80.0/256.0.

4.1.3 Dilate & Erode Image

The image processing unit's following phase dilates the binary image produced by the previous step using MATLAB's *imdilate* algorithm [31]. This method dilates grayscale, binary, or packed binary images by parsing a structuring element object, or array of structuring element objects, as returned by MATLAB's *strel* function as the second input. The image processing unit uses *square* as the structuring element object. Additionally, it makes use of the breakdown of a structuring element object. The resulting amplified image is further processed using another approach called erosion.

This phase erodes the image expanded in the previous phase. It uses the *imerode* algorithm from the image processing toolbox of MATLAB [32]. This function takes the same arguments as the *imidylate* method, but it uses *square* as the struct element object. As the next stage, image processing unit expands the eroded image from the previous procedure using the previously mentioned method. This stage results in HTML components being recognized and a smoothly expanded image.

4.1.4 Images of Element Highlighted

Finally, the related components in the binary image produced by the preceding steps are discovered. Then it transfers to a new image with random color squares. The *bwconncomp* technique from MATLAB's image processing toolbox [33] was utilized for this final stage. The final output of this image processing, made by drawing random color boxes, was used in the following stages of the proposed approach to convert the images into source codes.

4.1.5 Locating Detected Squares on Image

The number 2 section of Figure 7 depicts the unit that obtains the positions of each element as coordination values of detected squares. It generates a JSON file with the coordination values of each element. Furthermore, the same JSON file is

generated based on the hierarchy of the discovered element squares. It used a Python script to generate the structured JSON file, then MATLAB read the processed image with randomly colored images from the image processing unit. Figure 7 depicts an example of a simple hierarchy tree created using a simple UI.

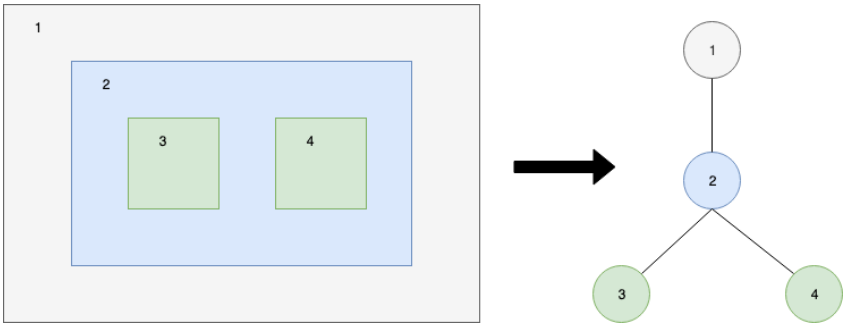


Figure 7: Unknown Tag Hierarchy of a Simple Input

The tag hierarchy is created in parallel with the placement of squares. The same JSON file is generated using this structure of elements. This procedure of generating the hierarchical tree of items made advantage of the Breadth First Search (BFS) traversal. Figure 7 depicts how BFS uses the hierarchy tree to determine the order of tag items.

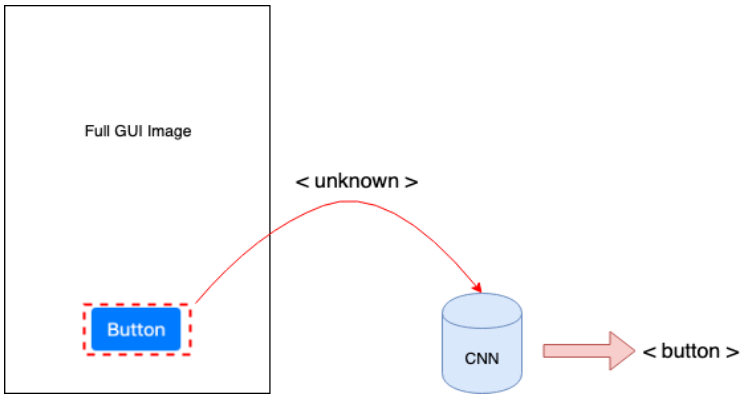


Figure 8: Unknown Tags Identify by CNN Module

Furthermore, this section inputs the overall GUI image and assigns coordinate values to each highlighted element. Figure 8 illustrates the HTML element detection flow in detail. In the first stage, it selects a specific element from the entire image based on the coordinate values and determines the name of the HTML tag corresponding to that element. The output JSON file is then updated with the actual HTML tag value rather than the unknown tag. Finally, the hierarchy tree of unknown tags is updated to an HTML tag hierarchy tree. Figure 9 summarizes the main picture.

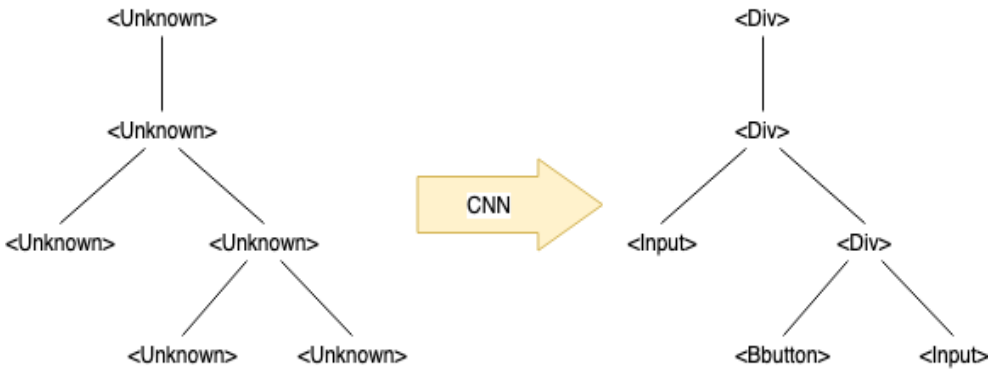


Figure 9: Use of Convolutional Neural Network

4.2. CNN Module

We implemented the HTML tag recognition module using *TensorFlow* [34] and *Python* technology. The module was trained on a *TensorFlow* environment hosted locally on a Mac (Linux-based) system, using *Python* scripts implemented using the *PyCharm* tool [35]. The training dataset for HTML tags is created using internet resources and human collection. Recently, the module was hosted in a cloud environment to train the CNN further and improve the prediction accuracy.

5. RESEARCH EVALUATION

In this study, two different modules are evaluated. They are an image processing module and a deep learning module. They are tested independently. The following scenarios are considered in evaluating the modules.

5.1 Image Processing Module

The image processing module was developed using the MATLAB Image Processing Toolkit. Overall, this module can output a source code file containing only unknown tags from the intermediate hierarchy tree. The input GUI image was evaluated by comparing the number of unknown tags found by the image processing module to the number of actual HTML tags in the source code. By comparing the input GUI image with the output image, which consists of random colored squares, the human eye can detect significant HTML element recognition errors. Figure 11 illustrates the results for the input GUI image in Figure 10.

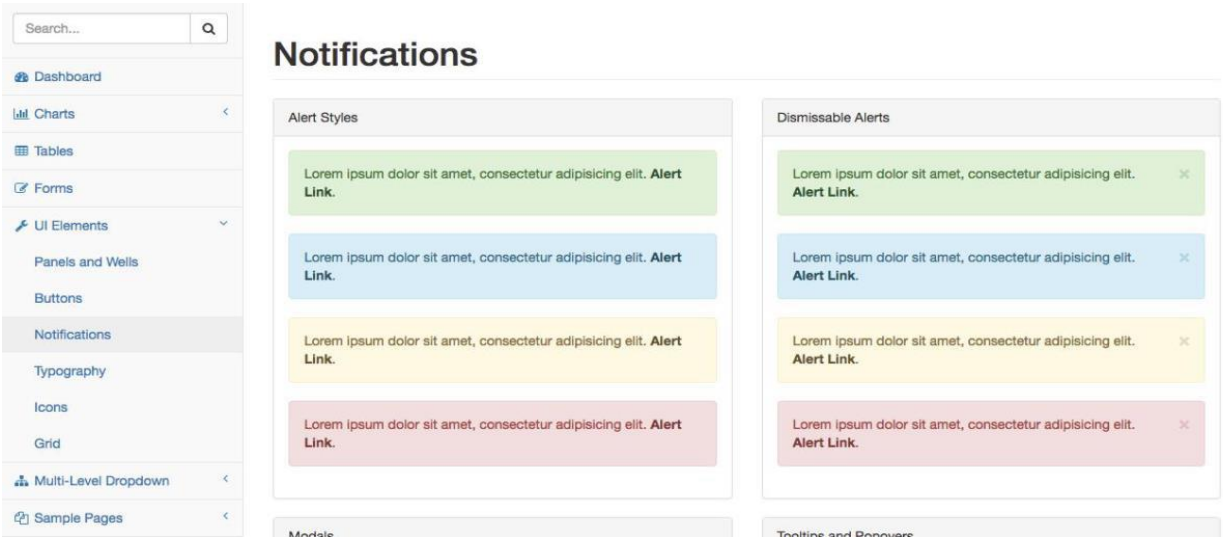


Figure 10: A Sample Input to Image Processing Module

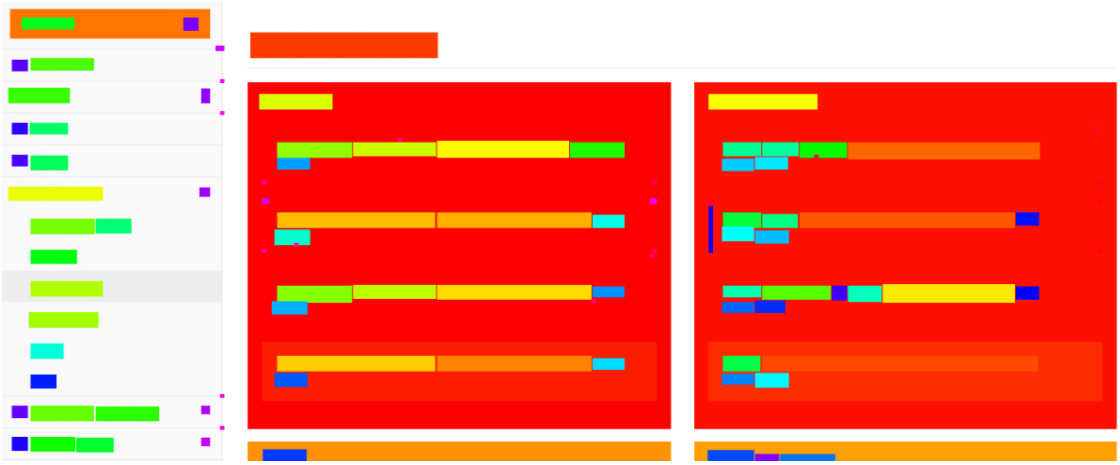


Figure 11: Sample of Generated Output Image

To evaluate the image processing module, we used multiple GUI images containing real HTML source code. The missing tag counts were then compared with the available source code created by a developer, and the unknown tag counts were obtained from the output picture of the image processing module.

5.2. Deep Learning Module

To evaluate the deep learning module, we used a comparative method using the same HTML source code. We compared the tag names in the original HTML source code and the modified one generated by the CNN module. Since the CNN was only trained to recognize a few types of components, we had to extract the recognized elements from the CNN module and match them to the corresponding elements provided by the developer for the input GUI image. We concentrated solely on the trained elements to logically evaluate the deep learning module.

To evaluate the results, a demonstration was conducted as described here. A simple one-page web page was created by hand using HTML and a screenshot of it was initially captured, as shown in Figure 12.

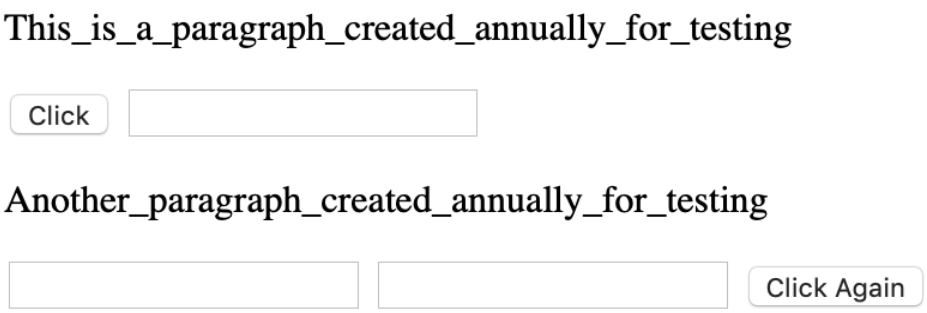
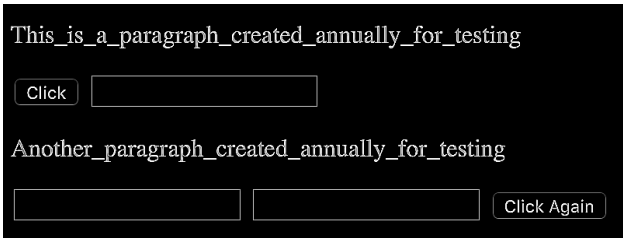


Figure 12: Sample Basic Web Page Created from Code

The screen capture was then sent via the image processing module to extract the corresponding HTML elements. Figures 13 and 14 illustrate the results of the first and last stages of the image processing module.



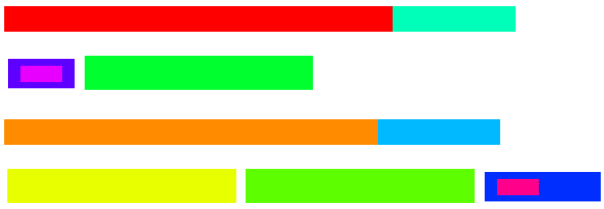


Figure 13: Image After First Step of Image Processing Module *Figure 14: Image After Completing All the Steps in Image Processing Module*

The detected area elements are then added to the training CNN hosted on AWS and predicts the HTML code as shown in Figure 15.

```
<html>

<body>
    <p></p>
    <button></button>
    <input />

    <p></p>
    <input />
    <input />
    <button></button>
</body>

</html>
```

Figure 15: Output code from the CNN

6. FUTURE DIRECTIONS AND CONCLUSIONS

The aim of the study is to implement a method to automatically generate source code from GUI images. Based on the qualitative analysis of the finally detected HTML tags, it can be concluded that the accuracy of the output of the image processing module affects the final desired result of the deep learning module. After analyzing the results of comparing the number of unknown tags with the number of expected real HTML tags, the image processing step should be further improved to extract HTML elements from the GUI image.

CNN was trained for only ten main HTML elements (heading, paragraph, image, hyperlink, button, text field, text area, icon, search bar, and label). According to the comparison between the results generated by the CNN module and the HTML source code that generated the experimental GUI images, the CNN is more accurate with the training data set. To improve the overall design of the proposed solution, attention should be paid to improving the image processing module rather than the deep learning module. To detect the highest number of HTML elements from the input GUI image, it was concluded that we need to improve the image processing module, as the CNN only retrieves elements extracted from the GUI image.

The design and evaluation approach of the proposed solution has been able to initiate an open discussion. The approach of first extracting HTML elements from the GUI and secondarily through deep learning and generating source code by processing BFS on a hierarchical tree should receive more attention from the research community. As a future work to improve this design, researching and extracting website taxonomy can provide source code to generate UI, improve prediction of HTML tags and even give suggestions to evolve GUI implementations. Since the research only focused on HTML tags instead of styling, improvements are needed by integrating styles into the identified elements.

ACKNOWLEDGEMENT

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

REFERENCES

[1] Myers, B. A. (1993). *Why are human-computer interfaces difficult to design and implement?* Carnegie-Mellon University. Department of Computer Science.

[2] Kumari, P., & Nandal, R. (2017). *A Research Paper on Website Development Optimization Using Xampp/PHP*. International Journal of Advanced Research in Computer Science, 8(5).

[3] Stephenson, N. (1999). *In the beginning... was the command line* (pp. 1-60). New York: Avon Books.

[4] Graphical user interface – Wikipedia: https://en.wikipedia.org/wiki/Graphical_user_interface

[5] Faure, D., & Vanderdonckt, J. (2010, June). *User interface extensible markup language*. In Proceedings of the 2nd ACM SIGCHI symposium on Engineering interactive computing systems (pp. 361-362).

[6] Coombs, J. H., Renear, A. H., & DeRose, S. J. (1987). *Markup systems and the future of scholarly text processing*. Communications of the ACM, 30(11), 933-947.

[7] Raggett, D., Le Hors, A., & Jacobs, I. (1997). *HTML 4.01 Specification*. IETF HTML WG.

[8] Berners-Lee, T., & Connolly, D. (1995). *Hypertext markup language-2.0* (No. rfc1866).

[9] Pressman, R. S. (2005). *Software Engineering: a practitioner's approach*. Pressman and Associates.

[10] Khaddam, I., Barakat, H., & Vanderdonckt, J. (2016). *Enactment of User Interface Development Methods in Software Life Cycles*. In RoCHI (pp. 26-35).

[11] Coda, F., Ghezzi, C., Vigna, G., & Garzotto, F. (1998, April). *Towards a software engineering approach to web site development*. In Proceedings Ninth International Workshop on Software Specification and Design (pp. 8-17). IEEE.

[12] 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).

[13] Murugesan, S., Deshpande, Y., Hansen, S., & Ginige, A. (2001). *Web engineering: A new discipline for development of web-based systems*. Web engineering: Managing diversity and complexity of Web application development, 3-13.

[14] Myers, B. A. (2004). 51. *Graphical User Interface Programming*. Graphical User Interface Programming.

[15] McFarland, D. S. (2007). *Dreamweaver CS3: The Missing Manual*. " O'Reilly Media, Inc.".

[16] Deng, Y., Kanervisto, A., Ling, J., & Rush, A. M. (2017, July). *Image-to-markup generation with coarse-to-fine attention*. In International Conference on Machine Learning (pp. 980-989). PMLR.

[17] Davody, A., Davoudi, H., Baba, M. S., & Florian, R. V. (2018). *Learning to generate HTML code from images with no supervisory data*.

[18] Balog, M., Gaunt, A. L., Brockschmidt, M., Nowozin, S., & Tarlow, D. (2016). *Deepcoder: Learning to write programs*. arXiv preprint arXiv:1611.01989.

[19] Gaunt, A. L., Brockschmidt, M., Singh, R., Kushman, N., Kohli, P., Taylor, J., & Tarlow, D. (2016). *Terpret: A probabilistic programming language for program induction*. arXiv preprint arXiv:1608.04428.

[20] Ling, W., Grefenstette, E., Hermann, K. M., Kočiský, T., Senior, A., Wang, F., & Blunsom, P. (2016). *Latent predictor networks for code generation*. arXiv preprint arXiv:1603.06744.

[21] Nguyen, T. A., & Csallner, C. (2015, November). *Reverse engineering mobile application user interfaces with REMAUI (t)*. In 2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE) (pp. 248-259). IEEE.

[22] Bahdanau, D. (2014). *Neural machine translation by jointly learning to align and translate*. arXiv preprint arXiv:1409.0473.

[23] Xu, K. (2015). *Show, attend and tell: Neural image caption generation with visual attention*. arXiv preprint arXiv:1502.03044.

[24] Sotak Jr, G. E., & Boyer, K. L. (1989). *The Laplacian-of-Gaussian kernel: a formal analysis and design procedure for fast, accurate convolution and full-frame output*. Computer vision, graphics, and image processing, 48(2), 147-189.

[25] Mallick, A., Roy, S., Chaudhuri, S. S., & Roy, S. (2014, January). *Optimization of Laplace of Gaussian (LoG) filter for enhanced edge detection: a new approach*. In Proceedings of the 2014 International Conference on Control, Instrumentation, Energy and Communication (CIEC) (pp. 658-661). IEEE.

[26] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). *Imagenet classification with deep convolutional neural networks*. Advances in neural information processing systems, 25.

[27] Sermanet, P. (2013). *Overfeat: Integrated Recognition, Localization and Detection Using Convolutional networks*. arXiv preprint arXiv:1312.6229.

[28] Johanna Pingel, *Edge Detection with MATLAB*. MathWorks: https://in.mathworks.com/help/images/edge-detection.html

[29] Huang, M., Mu, Z., Zeng, H., & Huang, H. (2015). *A Novel Approach for Interest Point Detection via Laplacian-of-Bilateral Filter*. Journal of Sensors, 2015(1), 685154.

[30] The MathWorks, Inc. *Image Processing Toolbox - im2bw:* https://in.mathworks.com/help/images/ref/im2bw.html

[31] The MathWorks, Inc. Image Processing Toolbox – imdilate: https://in.mathworks.com/help/images/ref/imdilate.html

[32] The MathWorks, Inc. Image Processing Toolbox – imerode: https://www.mathworks.com/help/images/ref/imerode.html

[33] The MathWorks, Inc, Image Processing Toolbox – bwconncomp: https://www.mathworks.com/help/images/ref/bwconncomp.html

[34] Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., ... & Zheng, X. (2016). *TensorFlow: a system for Large-Scale machine learning*. In 12th USENIX symposium on operating systems design and implementation (OSDI 16) (pp. 265-283).

[35] Shunan Zhang. Installing tensorflow on Pycharm (Mac). Stackoverflow: https://stackoverflow.com/questions/36998018/installing-tensorflow-on-pycharm-mac

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.