# Generating Website Codes from Images

Lai Nguyen Ha My[1,2], Huynh Ngoc Thien[1,2], Nguyen Duc Dung[1,2,*]
[1]Faculty of Computer Science and Engineering, Ho Chi Minh City University of Technology (HCMUT),
268 Ly Thuong Kiet Street, District 10, Ho Chi Minh City, Vietnam
[2]Vietnam National University Ho Chi Minh City,
Linh Trung Ward, Thu Duc District, Ho Chi Minh City, Vietnam
[*]Corresponding author: nddung@hcmut.edu.vn

## ABSTRACT

**Brainstorm website layout ideas usually start with everyone giving out their mockups, and all the team members will discuss to finalize the layout of the user interface. Once a vision of that mockup is accepted, it is given to the designer to sketch it digitally on computer software (i.e., Photoshop, Figma, Sketch). When the designer completes, the developer based on the final design to code the UI/UX of the website. As we can see, the process requires three stages, which can be time-consuming. Therefore, if anyone has an idea for the professional website layout, they can visualize it by drawing on sketches. However, it can be impossible for them to make a usable website without designers and website developers. Due to that reason, our primary goal in this paper is to help individuals transform their hand-drawn sketch images into a website that can be deployed. To achieve that goal, we present two approaches: classical computer vision techniques and the other using a deep learning model to detect the sketch and execute the conversion. Furthermore, our evaluation shows that deep learning is the most promising direction. Still, classical techniques also improve the model's input data by applying it in the pre-processing image.**

***Index Terms*—Keywords: Computer vision, Hand-drawn sketch, UI/UX, Deep learning**

## I. INTRODUCTION

By sketching a mockup on paper to illustrate the structure of the user interface, we face a challenge that designers need to convert their mockup into the design, which is a type of scripted language. After finishing, they need to pass the design to a developer and having the developer implement another scripted language (HTML with CSS, JS) to turn the design into an actual website. This process contains conversion between two different languages, which costs much time in project development.

Solutions to shorten the process had been proposed by several companies such as Microsoft, Figma, Uizard. For instance, when using the Figma[1] to cut down the process on half by removing the developer's involvement with designs to code conversion. Moreover, **SILK** [1] turns digital drawings into an application code using gestures; **DENIM** [2] augments drawings to add interaction, and **REMAUI** [3] convert high-fidelity screenshots into mobile apps.

Those above-mentioned applications rely on classical computer vision techniques and mathematical conversion to achieve the following goals:

- Faster process - Mock-ups converted directly to a design or a source code without human interference.
- Less work - Developers can shorten the UI process - coding and focus on improving features of the website.

However, they cannot fully get what they aim for since the classical techniques with mathematical conversions have its flaw and limitation. Furthermore, while pointing out drawbacks of classical means, we figure out that we can borrow the existing Deep Learning techniques that already proved their efficiency to apply and achieve significant results if we treat this kind of problem as object detection.

In this paper, the main contributions of this work are outlined as follows (shown as Fig. 1):

- Build both a deep learning and classical computer vision means which can:
  - Detect and classify website elements sketched on the paper and subsequent tasks (i.e., line and text detection).
  - Adjust the layout to fit the defined rules for a beautiful website layout.
  - Display the detection results and return them as a response for API service.
  - Convert the detection's result to a website source code.
- Evaluate the performance of classical techniques and the Deep Learning model.

The rest of this paper is organized as follows: in the next section II presents an overview and discussion for related works. In Section III, we show our methods, our approaches, the pre-processing steps conducted. In Section IV, we present our experimental results and discussion. To sum up this study, Section V presents conclusions and future works.

## II. RELATED WORK

In this section, we discuss survey literature related to the processing and analysis of converting drawings to a website. This also will be followed by a summary of the algorithm using to detect the hand-drawn sketch and markers to recognize the website component.

### A. Sketch2HTML of Microsoft

Microsoft solves this problem by detecting the boundary shape to recognize website components and the text inside

[1]Anima for Figma, https://www.animaapp.com/.

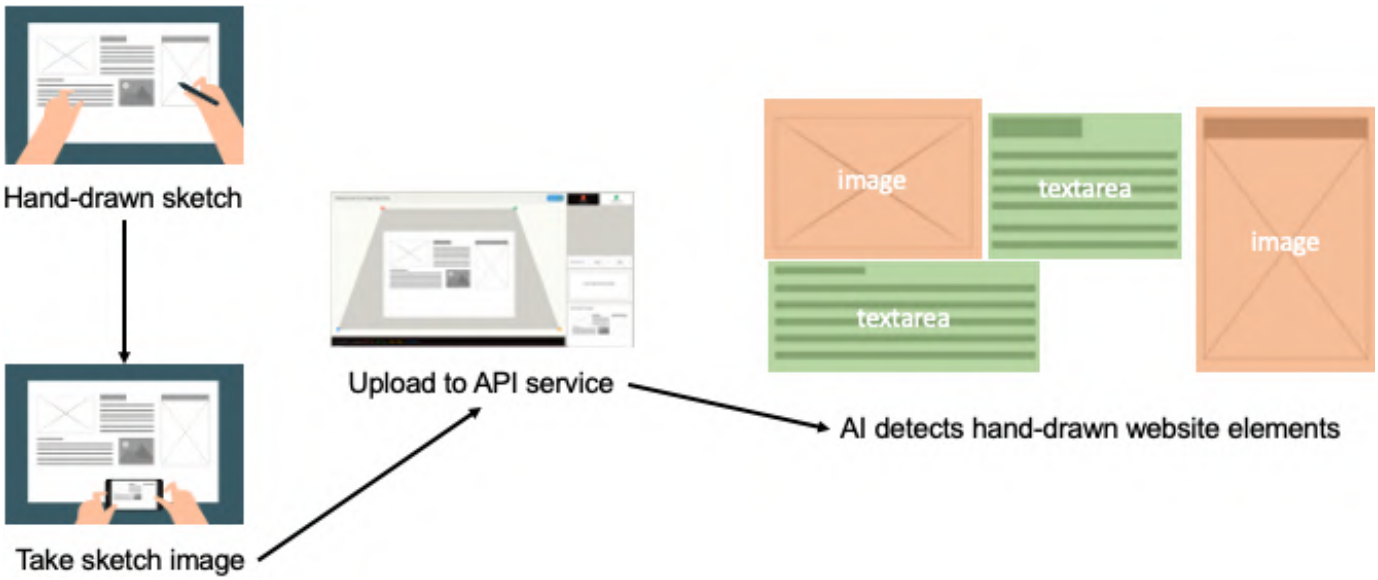


Fig. 1: The goal we achieve in this paper.

each element to generate tag text for the layout. Moreover, they also add symbols to the regular rectangle shape to classify website components [4]. However, they only focus on building it as a beta project since the number of drawing website elements supported to be detected are not many. The reason for that is the challenge of similarity between elements, which is presented in Fig. 2.

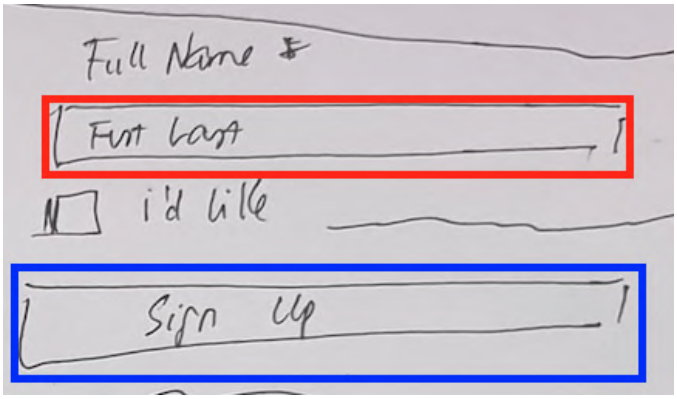


Fig. 2: The element in the red rectangle is drawn as input, while the blue one is marked as a button. Moreover, the two elements are drawn precisely the same way.

### B. Pix2Code and other similar products

The following product we examine is Pix2Code. This source code project on GitHub has belonged to Uizard product based on the following paper [5]. This method of the paper only focuses on comparing GUI components (which is already markup code for UI components) with input images to see if they are matched or not. If they are matched, then they are classified as predefined GUI components. Due to this reason, the paper had claimed this model would not work for the real-life conversion of the application layout. However, it applies the Convolutional Neural Network (CNN) method, which we are also interested in detecting symbols, so it is still useful when providing some techniques mentioned by the paper. Moreover, the following source codes are also based on Pix2Code above. The exciting thing is the method [6] they use that may serve the future purpose - extending this project not only detecting hand-drawn sketches but also GUI components and website images screenshot by users' computers.

- Sketch2HTML[2]
- SketchCode[3]

### C. Generation of Slides from Hand-Drawn Sketches

In this paper [7], the author describes an algorithm to detect basic hand-drawn shape images taken by a handheld device. Fig. 3 shows how the algorithm works.

We figure out that by removing corners, the algorithm can avoid the situation when users draw two lines intersecting each other, and they overdraw the line a bit (which can make the computer system misunderstand that there are two corners as one is the intersection points and the other is the endpoint of the overdraw line). Moreover, the Warp Perspective of the input image is a good way to ensure the line detection since images are taken by handheld devices, and they are not always straight up to the camera. In addition, a predefined list of shapes can help the algorithm understand what to base on and detect, which means save a lot of time training for learning shapes if the method is based on Deep Learning. However, the author did not clearly mention how they reduce noise from input images since all inputs in the result seem to be in really perfect conditions, which is not true if one person takes an image in real life, therefore lead to the situation that this experiment's algorithms may not work for real-life using. Besides, the hand-drawn image shows that all the lines are

[2]iSysLab. GitHub: github.com/iSysLab/sketch2html.
[3]Ashnkumar. GitHub: github.com/ashnkumar/sketch-code.

perfectly drawn, which is not ideal in real life since there is a situation that a human's hand can mess up a drawing.

### D. Sketch2Tag: Automatic Hand-Drawn Sketch Recognition

Most methods use a particular domain (shape characteristic) or the predefined list of shapes for detection. However, this paper [8] uses another approach which is heavily based on existed clip art of the shapes and the probabilities of users vote while using the system. For example, users draw a part of the shape, and by looking at the recommended system, they found out their shape in need and choose it, which makes the part of the shape they are drawing will be used as the template matching for the next time other users draw. The fascinating thing in this paper is that the system has users involving in recommending what kind of shape style they usually draw, which can help increase the accuracy of the system learning algorithm. Moreover, the correct results do not need to be defined by the system since if the result of shape detection is agreed and chosen by most users, and it is eventually the correct answer that everyone wants. In contrast, if the number of users uses this tool is not much, the accuracy will never be improved since the result is only correct when the users are satisfied. In addition, the website layout in our project, the website components that need to be drawn are a lot and combine or overlay next to each other, which is hard to create enormous resources from the beginning for the recommendation system.

## III. Proposed System

Most drawings contain a symbol used to detect the website component, connectivity information (lines), and some form of annotation (text). However, there is no public dataset available for evaluation purposes. In section III-A, we apply the classic approach for the detection drawings. The following subsection III-B will discuss in detail the modern means and its experiments. This will include data exploration and how we pre-process the data.

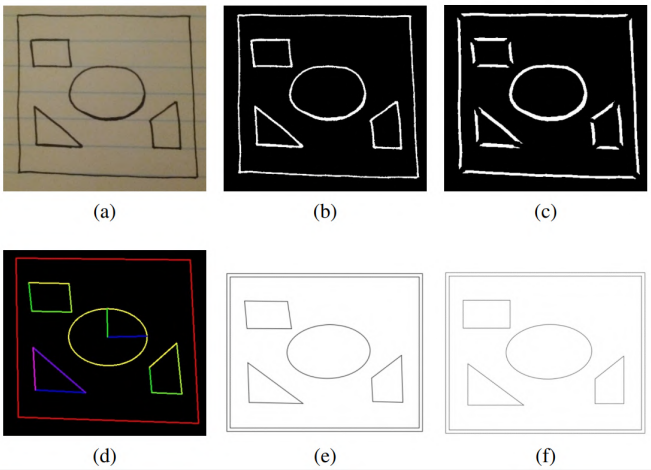


Fig. 3: How the algorithm works.

### A. Classical approach

After referencing the processing way of providing solutions from several papers and companies, we propose a better version to detect hand-drawn website elements in sketch images.

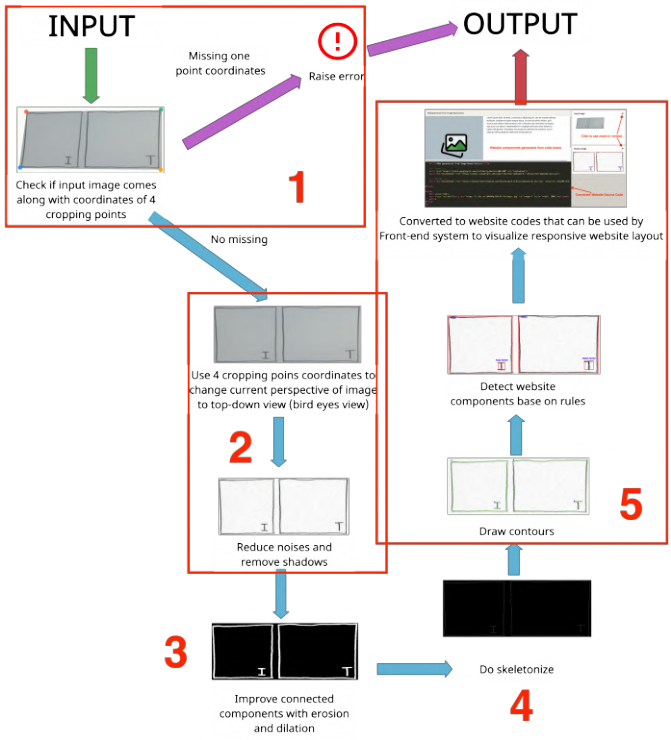


Fig. 4: An entire process of the classical approach

Following Fig. 4, we explain each process and its scopes in order to generate the final detection result.

1) Every input image that we detect must be sent to the API with the image source file and its four corners coordinates (If any conditions are missing, the API will raise and response error). These coordination tend to help the algorithm know where to align the top view using Warp Perspective Algorithm [6].
2) Images need to have less noise in order not to affect the algorithm performance, so we designed a technique to remove the shadow out of the input image. This technique is called bitwise removal. A brief explanation about this means, firstly, we dilate the image to make the content in the sketches (hand-drawn website component layout) blur so that the shadow and the background seem to be the same. After that, we compare the absolute difference between the new blur image and the old one. In the content area, by using the dilation, the new image and the old one tend to be different, which causes the pixel to be 1 (binary image). For other areas such as background and shadow zone, because the blur makes them similar to each other, then the comparison tends to be equal, and the pixel has 0, which helps to remove the shadow since it is blended into the background. The final step is inverting the pixel value, and by doing that, we are able to delete both background and shadow. After removing the noise and shadow, drawing lines and

shapes by users may miss some connections in someplace where the shadow was removed. Therefore, we apply the Morphological method to dilate and thicken drawing lines in order to connect them as one so that they can form the shape that the user drew in the first place [6].

3) Thickened lines that are created to form the shape can cause a redundant contour since the contour only detects the edge of a line, which if it is too thin, it tends to create an inner line and outer line. Therefore, we applied another means called Skeletonize [9] to get the result.
4) When the hand-drawn lines are thin, by using contour, we can draw lines that form as regions of elements and know which part is drawn as a rectangle shape by approximating the polygons at the corner of each shape (which we call endpoints). And then, based on the number of polygons that had been approximated, along with our defined rules for each polygon number, we can detect the website elements based on the shape region with a specific symbol inside. For instance, symbols in the rectangle are drawn as an I, it maps onto Image element, and similarly, drawn as T, it maps onto Textarea element to convert to a website source code.

### B. Modern approach

There are several object detection approaches that be listed below:

- Fast R-CNN
- Faster R-CNN
- Mask R-CNN
- Single Shot MultiBox Detector (SSD)
- YOLO (You Only Look Once)

After researching and experimenting with the example source code with the pre-trained model of each technique, we consider between two pre-trained models: SSD and Faster RCNN.

*1) SSD*

SSD might appropriate for this problem because:

- SSD pre-initializes the boxes at each position on the image, computes and evaluates information at each location to see if there is an object or not, which object is, and based on results of the close proximity, the SSD tends to calculate a box that covers the object.
- SSD calculates the bounding box on different feature maps, and at each feature map layer, a box tends to embrace the image with different dimensions partially. By using multiple boxes on multiple floors, SSD can aggregate the resulting box size and position, and with excluding Region Proposal Network (RPN), SSD can also achieve a higher processing speed.

However, we decide not to choose SSD because its lightweight may help us achieve the speed but not the accuracy performance we desire because the region proposal layer has been removed. Furthermore, due to that fact, Faster RCNN is another choice that we finally come to confirm using.

*2) Faster RCNN*

The Faster RCNN model is chosen for these reasons:

- Providing a new layer called ROI (Region Of Interest) Pooling that extracts equal-length feature vectors from all proposals in the same image.
- Building a network that has only a single stage.
- Sharing computations (convolutional layer calculations) across all proposals rather than independently calculating each proposal. This is done by using the new ROI Pooling layer.
- Not caching extracted features and thus does not need so much disk storage.
- Fast enough without asking to trade off accuracy.

After choosing the model for Deep Learning techniques, we do some more research about training sets and found out that most hand-drawn sketches tend to be drawn on white paper. From a user's perspective, he/she can draw on any platforms he/she like; however, the platform must be a single color, or the background must have less noise, such as grid or dot in a notebook as Fig. 5:

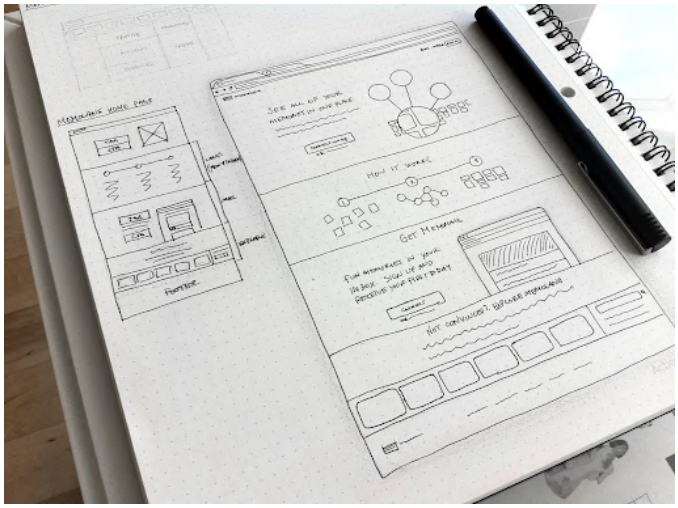

Fig. 5: Notebook with dot background

Now, the novelty point we apply to the existing pre-trained model is separating the input training set into three categories as binary, grayscale and raw in other to figure out with which dataset, the model will work best and have the promising result (the input image does not contain the grid background):

Following Fig. 6, raw images and their two conversion types - binary and grayscale, will be labeled separately and trained on Faster RCNN. The final validation result of the model reveals that with which kind of dataset, the model has a better detection result. From there, we can evaluate how Deep Learning in the modern approach tends to solve this problem in comparison to the classical approach.

## IV. Experimental Results

For each experiment on each approach, we develop or defines rules to normalize the input and output in order to have one metric to evaluate the performance and accuracy.

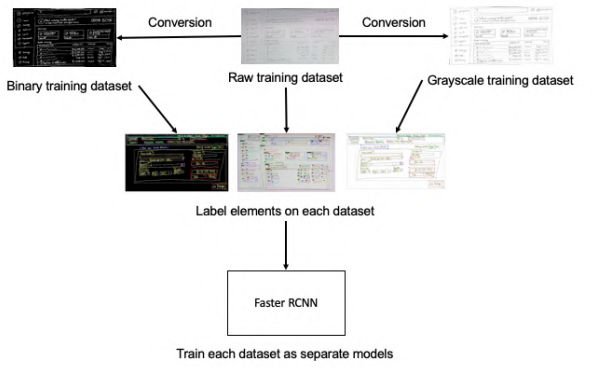


Fig. 6: Steps to train the custom dataset on the pre-trained Faster RCNN model.

### A. Detection results from the classic approach

For the classical approach, along with an image process, we also includes a web application to receive the detection result and convert it to source code. By doing this, we can know the speed when using the classical approach to detect hand-drawn sketches.

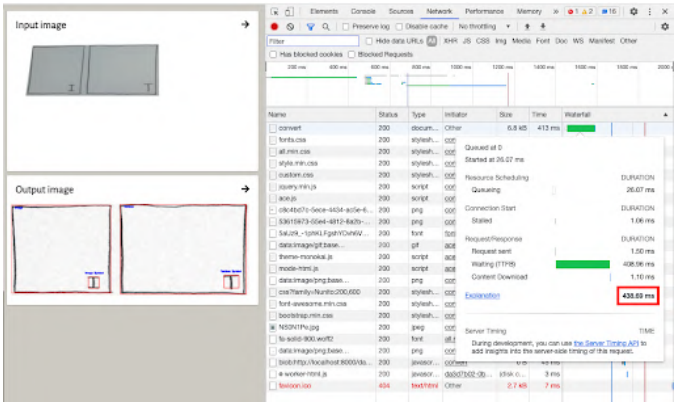


Fig. 7: An example that we build to test the experiment of detecting and converting to HTML source code, which only costs about 438.69 ms (excludes all other web-app loading)

Moreover, for the result in Fig. 7, the time for the completion tends to be less than 1 second when it is only based on a mathematical conversion. Even though, the speed is fast, and the result is acceptable, there is one problem that we cannot cope with is extending more elements since some symbols tend to have the same endpoints (such as L and U both have two endpoints).

Therefore, we had taken some parts of the classical approach to improve the image's pre-processing step of the modern approach to boost the model detection with a better result (we use the classical approach to remove noises and shadow).

### B. Detection results from modern methods

By applying the classical approach in the pre-processing image, we can ensure that the input dataset for the Faster RCNN model tends to be clean and have all the features for the model to extract and recognize. In addition, to evaluate the model with three different training datasets, we need to scope out the number of label elements we use to train and detect (there will be 20 label elements) and the training iteration that we expect the convergence will occur (which we set default as 4000 iterations).

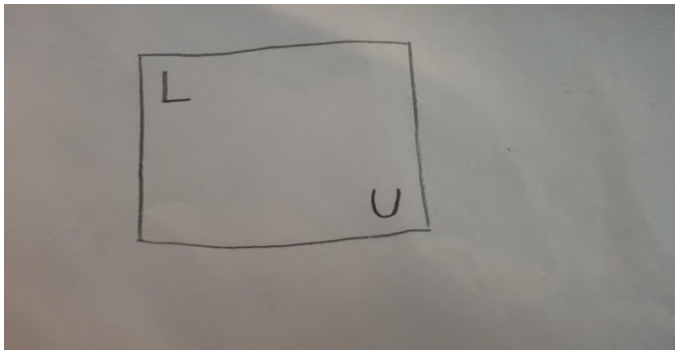


Fig. 8: Case where classical approach will confuse between two symbols

With all the above-mentioned setup, after training the model and run through the validation dataset (both raw images from several individuals and augmented images from the existing dataset), we retrieve a table of the average precision, which is also the standard metric to evaluate three different models derive from three different image types.

In the table I, the unit is on a 100 percent scale where the higher the number, the better the precision in detecting each

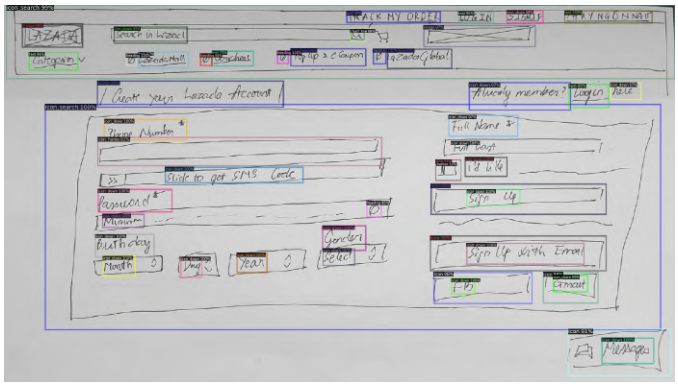

Fig. 9: The result of the model with the Raw dataset

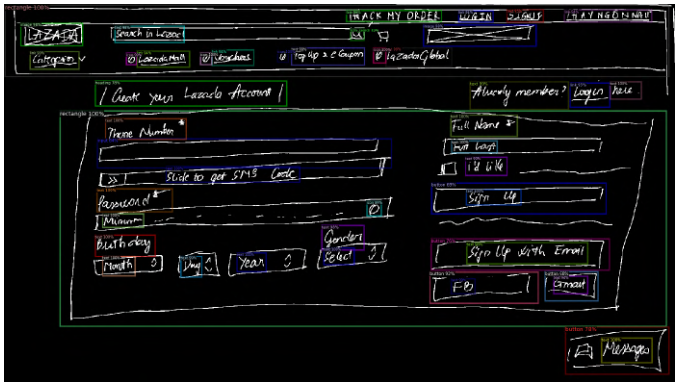

Fig. 10: The result of the model with the Binary dataset

| Label Element | Raw Dataset Average Precision | Grayscale Dataset Average Precision | Binary Dataset Average Precision |
|---|---|---|---|
| image | 80.944 | 75.154 | 51.676 |
| heading | 100.000 | 88.570 | 42.933 |
| rectangle | 91.089 | 82.325 | 62.606 |
| icon_search | 80.198 | 79.059 | 0.000 |
| icon_menu_horizontal | 61.683 | 45.956 | 0.000 |
| avatar | 90.231 | 88.317 | 12.475 |
| image_round | 0.000 | 0.000 | 0.000 |
| link | 96.287 | 95.465 | 28.436 |
| icon | 84.836 | 83.121 | 66.428 |
| checkbox | 80.000 | 60.198 | 0.000 |
| input | 70.000 | 80.099 | 0.000 |
| icon_close | 80.000 | 0.000 | 0.000 |
| icon_down | 80.231 | 62.566 | 28.614 |
| input_search | 70.000 | 0.000 | 0.000 |
| text | 89.239 | 73.052 | 69.193 |
| divider | 54.919 | 24.079 | 0.000 |
| button | 83.843 | 82.122 | 52.941 |
| icon_right_arrow | 80.000 | 90.000 | 0.000 |
| icon_plus | 81.683 | 41.909 | 0.000 |
| icon_home | 65.050 | 86.634 | 0.000 |
| chart_line | 0.000 | 0.000 | 0.000 |

TABLE I: Performance table of each detected element in every Faster RCNN model

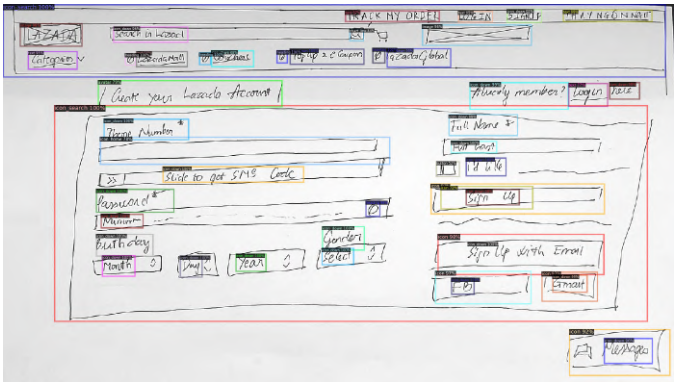

Fig. 11: The result of the model with the Grayscale dataset

element. Furthermore, we can see that the number of elements that got detected correctly (high percentages) is very high in the training model on the Raw and Grayscale datasets. While it is deficient or could not get high rates when training with the Binary dataset. However, Fig. 9, Fig. 10, and Fig. 11 show the detection result of each model; we can see that model with the Binary dataset is the only one with the highest elements got right while others predict all the elements with their wrong type. This gives we a confirmation that due to noises and similar features between elements (both drawn by the same color and lines), the model with the Raw and Grayscale model had been overfitted. These models try to map a label randomly to any element it can find because the similar features are huge, resulting from noises interference. On the other hand, the model with binary had been normalized to achieve the cleanest input since the background will always be 0 while the element will always be formed by 1 pixel.

Because of the result above, we can conclude that if we apply Deep Learning to solve this problem better, we also need to combine it with the classical approach in the image's pre-processing stage and convert it to the binary mode to achieve the most accurate result. In addition, when it comes to the significant result by applying binary dataset with Faster RCNN, we also base on that to provide a solution API service that can support third-party users (companies) to label their own elements in website sketches to train a detection model.

## V. Conclusion

By using the classical approach, we can detect very fast, and we can ensure that the detection results will have high accuracy since it is all based on mathematical conversion. However, this method cannot be extended if we keep detecting elements by endpoints, so we decided to combine it with a modern approach. On the other hand, deep learning of the modern approach proves itself a promising way to detect various elements if only those elements are labeled and trained. Due to that fact, we had already built a tool for third-party users to label their own elements and convert to the design or website source code they want base on the detection result

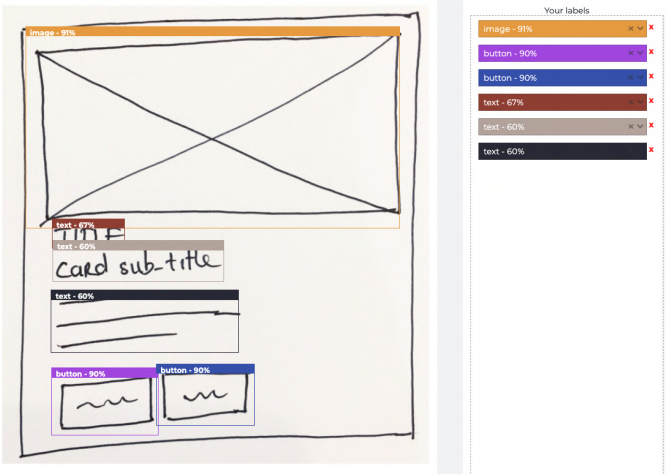


Fig. 12: A labeling tool for each software company uses its own rules

they wish to have when labeling on their own rules (shown as Fig. 12).

Overall, we had contributed many benefits for the detection solutions by trying the experiment in this paper, and therefore, we will continue to improve it by applying more complicated models such as hand-written OCR in order to generate more elements such as custom text on the sketch.

## ACKNOWLEDGMENT

This research is funded by the Office for International Study Programs (OISP), Ho Chi Minh City University of Technology (HCMUT), VNU-HCM under grant number SVOISP-2021-KH&KTMT-34. We acknowledge the support of time and facilities from HCMUT, VNU-HCM for this study.

## REFERENCES

[1] J. A. Landay and B. A. Myers, *Sketching interfaces: toward more human interface design. Computer 34.3, pages 56–64,* March 2001.

[2] James Lin et al., *DENIM: Finding a Tighter Fit Between Tools and Practice for Web Site Design. Computer 34.3, pages 510–517,* April 2000.

[3] 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), pages 248–259,* November 2015.

[4] R. Ingle, Yasuhisa Fujii, Thomas Deselaers, Jonathan Baccash, and Ashok Popat, *A scalable handwritten text recognition system. pages 17–24,* September 2019.

[5] Alex Robinson, *Sketch2code: Generating a website from a paper mockup* May 2019.

[6] Liao, Tianli, and Nan Li, *"Single-Perspective Warps in Natural Image Stitching."* ArXiv.org, 7 Mar. 2018, arxiv.org/abs/1802.04645.

[7] Beltramelli and Tony, *pix2code:Generating code from a graphical user interface screenshot.* arXiv preprint arXiv:1705.07962, 2017.

[8] Durjoy Sen Maitra, Ujjwal Bhattacharya, and Swapan Parui, *Cnn based common approach to handwritten character recognition of multiple scripts. pages 1021–1025,* August 2015.

[9] T. Y. Zhang and C. Y. Suen, *A fast parallel algorithm for thinning digital patterns.* Commun. ACM, 27:236–239, March 1984.

[10] Muneeb Ahmed and Jeff Wheeler, *Generation of slides from hand-drawn sketches.* 2014.

[11] Yuntian Deng, Anssi Kanervisto, and Alexander Rush, *What you get is what you see: A visual markup decompiler.* September 2016.

[12] L. Donati, S. Cesano, and A. Prati, *A complete hand-drawn sketch vectorization framework. Multimedia Tools and Applications, pages 1–31* February 2019.

[13] Graham Finlayson, Steven Hordley, and Mark Drew, *A fast parallel algorithm for thinning digital patterns.* Commun. ACM, 2353:129, 2002.

[14] Manuel J. Fonseca and Joaquim A. Jorge, *Using fuzzy logic to recognize geometric shapes interactively.* IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2:1–20, May 2000.

[15] Manuel J. Fonseca and Joaquim A. Jorge, *Multi-domain recognition of hand-drawn diagrams using hierarchical parsing.* Multimodal Technologies and Interaction, August 2020.

[16] L. Zhang Y. Rui J. Wu, C. Wang, *AAAI'14: Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence, page 951–957,* July 2014.

[17] Joaquim A. Jorge and Manuel J. Fonseca., *simple approach to recognise geometric shapes interactively. Graphics Recognition, Recent Advances, Third International Workshop,* GREC'99 Jaipur, India, September 26-27, 1999, Selected Papers, 1941:266–276, January 1999.

[18] Felipe Such, Dheeraj Peri, Frank Brockler, Paul Hutkowski, and Raymond Ptucha, *Fully convolutional networks for handwriting recognition.* July 2019.

[19] Zhenbang Sun, Changhu Wang, Liqing Zhang, and Lei Zhang, *Sketch2tag: Automatic hand-drawn sketch recognition. MM 2012 - Proceedings of the 20th ACM International Conference on Multimedia, pages 1255–1256,* October 2012.