Automatic HTML Code Generation from Mock-up Images Using Machine Learning Techniques

Batuhan Aşıroğlu
Computer Engineering Department
Istanbul University - Cerrahpasa
Istanbul, Turkey
b.asiroglu@ottoo.com

Büşra Rümeysa Mete
Computer Engineering Department
Istanbul Kultur University
Istanbul, Turkey
b.mete@iku.edu.tr

Eyyüp Yıldız
Computer Engineering Department
Erzincan Binali Yıldırım University
Erzincan, Turkey
eyyup.yildiz@erzincan.edu.tr

Yağız Nalçakan
Computer Engineering Department
Izmir Institute of Science and Technology
Izmir, Turkey
yagiznalcakan@gmail.com

Alper Sezen
Turkcell Teknoloji
Araştırma ve Geliştirme A.Ş.
Istanbul, Turkey
alper.sezen@turkcell.com.tr

Mustafa Dağtekin
Computer Engineering Department
Istanbul University - Cerrahpasa
Istanbul, Turkey
dagtekin@istanbul.edu.tr

Tolga Ensari
Computer Engineering Department
Istanbul University - Cerrahpasa
Istanbul, Turkey
ensari@istanbul.edu.tr

***Abstract*—The design cycle for a web site starts with creating mock-ups for individual web pages either by hand or using graphic design and specialized mock-up creation tools. The mock-up is then converted into structured HTML or similar markup code by software engineers. This process is usually repeated many more times until the desired template is created. In this study, our aim is to automate the code generation process from hand-drawn mock-ups. Hand drawn mock-ups are processed using computer vision techniques and subsequently some deep learning methods are used to implement the proposed system. Our system achieves 96% method accuracy and 73% validation accuracy.**

***Index Terms*—Object detection, object recognition, convolutional neural network, deep learning, automatic code generation, HTML**

I. Introduction

The importance of the Internet web sites has increased considerably due to the progress made in today's technology. Nowadays web sites reflect the face of the states, institutions, communities, people, etc. Web sites are available in almost every field, from knowledge to social work, from games to training and many more. Web sites created by companies come to the fore for financial reasons for product marketing or advertising purposes. On the other hand, official institutions aim to provide more efficient services.

At the front-end of every web site is a "web page" which is the part of the web site that interacts with the user. It is very important to serve a page that attracts the attention of the end-user, is easy to use and has enough functional features. However, developing web pages that respond efficiently to these needs involves a very burdensome process. In the preparation of web pages, graphic designers, software specialists, end-users, corporate authorities and people employed in many different areas are required to work together. Usually, the process starts with the mock-up design of the user interface by the graphic designers or mock-up artists, either on paper or a graphic editing software, in line with the needs of the institution. Software experts write code for the web pages based on these drafts. The resulting web pages may change based on feedback received by the end users. There process involves a lot of repetitive tasks. Rewriting the code for components with similar functions and page structures changing over time makes the process tedious. This reveals the need to explore more efficient solutions in web page design.

The idea of designing a web page by generating automatic code is gaining importance as a research subject. Automatic production of web pages reduces programming time, process cost and resource consumption. Thanks to the faster progressive design stages, the final web-site is produced in a shorter time.

In this study, an algorithm has been developed to automatically generate the HTML code for hand-drawn mock-up of a web page. It is aimed to recognize the components created in the mock-up drawing and to encode them according to the web page hierarchy. A public dataset of hand-drawn images of web sites obtained from Microsoft AI Labs' Github page [1] is used to train and verify the proposed scheme. The images on the dataset are processed using computer vision techniques and a deep neural network model involving convolutional neural networks is used to train the data. Afterwards a structured HTML code is obtained. Our model achieves 96% method accuracy and 73% validation accuracy.

978-1-7281-1013-4/19/$31.00

The rest of the paper is constructed as follows; Section II presents similar studies in the literature. Section III and IV describes the dataset used and the methods. In Section V, the results and findings obtained were shared. The Section VI, which is the conclusion section, includes evaluations.

II. Related Work

In [2] an algorithm dubbed as REMAUI finds the components of the user interface a mobile application such as buttons, textboxes, pictures and creates the code for them from the screenshots of an application window or conceptual drawings. In their study, which was the the first in terms of providing conversion to the code from the screen images or drawings for mobile platforms, computer vision and optical character recognition methods are used. Although the REMAUI method works successfully, it does not support cross-page transition and animations within the page. In [3] the authors developed the P2A algorithm to remedy the deficiencies of the REMAUI algorithm.

The authors in [4] developed the *pix2code* algorithm which aims to convert the graphical interface for a web page to structured code using deep learning with convolutional and recurrent neural networks. The method has been tested on Android, IOS and mobile platforms and successful results have been obtained.

In [5], the algorithm named ReDraw takes mock-ups of mobile application screens and creates a structured XML code for it. In the first stage of their implementation computer vision techniques are used to detect individual GUI components. The second stage involve classification of the detected components according to their function, e.g. toggle-button, text-area, etc. In this stage deep convolutional neural networks are used. In the last stage, the XML code is generated by combining with the kNN algorithm according to web programming hierarchy.

Nowadays open source code libraries such as Github [6] are used very frequently to share code and applications. It is a common practice to investigate this repository and reuse code when starting or improving software projects. The shared codes in these libraries reduce the same code being written over and over by different people. In [7], the authors use a search program called SUISE in which the users define a graphical interface with simple drawings and keywords. This interface is then searched in existing libraries to obtain similar interfaces. These interfaces are turned into operable codes and returned to the end user to select the most suitable interface.

Microsoft has recently developed a system that takes hand-drawn mock-ups of simple web pages and creates the structure HTML code [1]. There are no literature that explains their work, however they have published their code and dataset online. In this project we use some of the images from this dataset.

III. Dataset

In this study, we used some of the images provided from Microsoft AI Lab for their Sketch2Code application [1] in order to create our dataset. As we create it for the experiment, we selected the images that contains four type of components: textbox, dropdown, button, and checkbox. Then we cropped each component from these images and collected them to train our CNN model shown in Fig. 5.

IV. Methods

This study was carried out in four basic steps. In the first step, object detection was applied on the input image with image processing techniques such as erosion, dilation and contour detection. After this stage, the identified objects were cropped and then the components obtained were labeled with the trained CNN model. Finally, the output of this model has been converted to HTML code through the HTML Builder script. The algorithm proposed is visualized in Fig. 1.

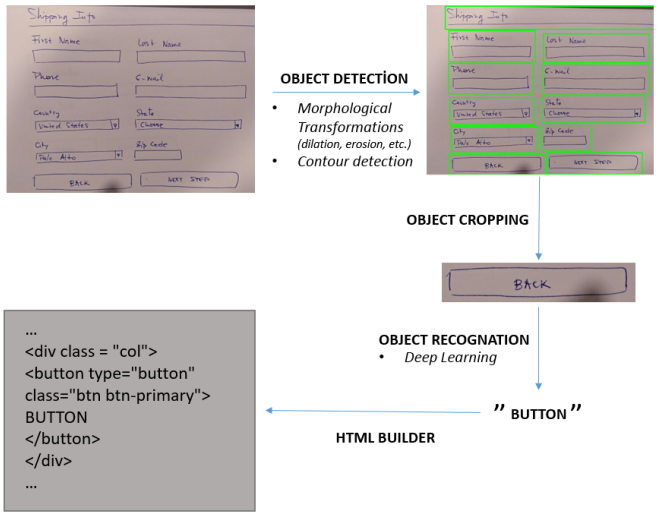

Fig. 1. Proposed algorithm

A. Object Detection and Cropping

After reading the input file, it is converted to gray scale format. Then, Gaussian Blur was applied 2 times to them with 3x3 rectangle kernel. After the threshold process was carried out, rectangle were drawn by applying the contour detection algorithm to determine the objects by applying morphological transformations. In this way, the components in the input image have been detected. The detected components were cropped to be transferred to the CNN model. The output of this stage is shown in Fig. 2 and Fig. 3.

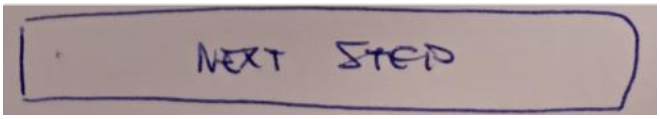

Fig. 2. The output of the cropping

In the stages of morphological transformations, 8 iteration dilation was performed with 4x4 rectangle kernel. Then erosion process were applied with 3x3 ellipse kernel for 4 iterations and dilation process was performed with 1x10

rectangle shaped kernel for 2 iterations . Finally, a 10x1 rectangle shaped kernel was used for dilation process. These operations are shown in Fig. 4

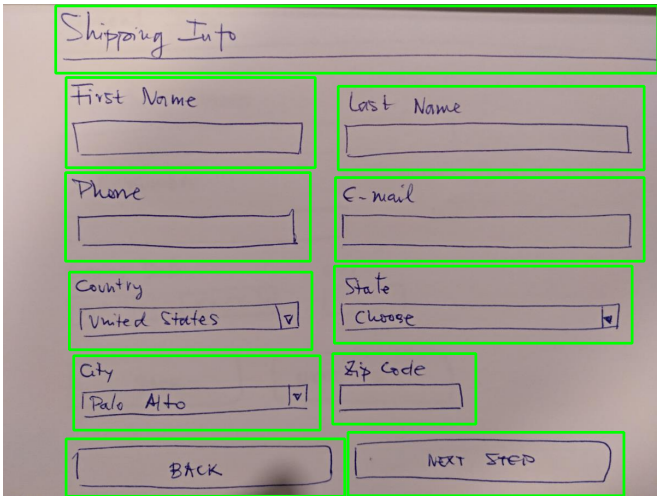

Fig. 3. The output of the object detection process

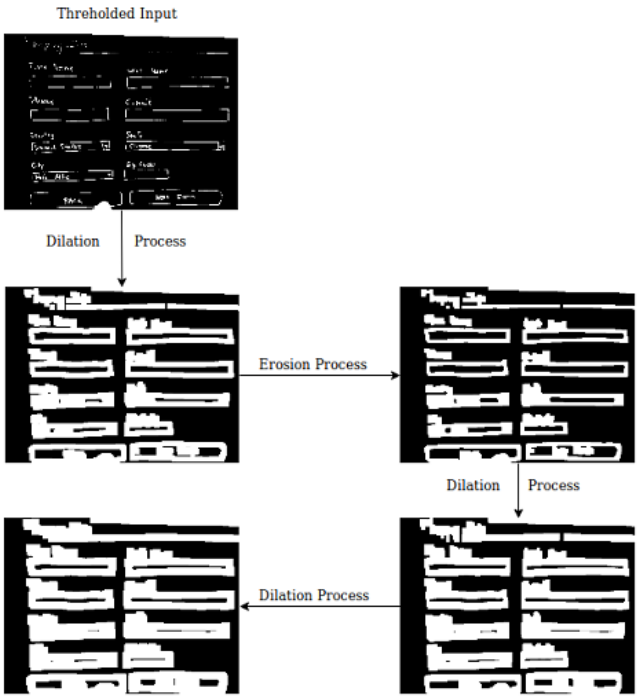

Fig. 4. Morphological transformation processes

B. Object Recognition

The model shown in Fig. 5 was trained with the elements in our component dataset. As mestioned before it consists of four different types of components such as textbox, dropdown, button and checkbox.

After the stage of training the model, the loss function was trained for 200 epochs using Binary Crossentropy and RMSProp algorithms by setting the batch size to 64. Afterwards, component recognition process was carried out by giving the cropped components that came from the previous stage as input.

As seen in our CNN Model in Fig. 5, we put several convolution layers with 4x4 kernels and then we applied max pooling processes with 2x2 kernels for the feature extraction purpose. After the process that we call as vectorization of the features we put BiLSTM layer for catching correlation of the extracted features. After all, we put Full Connected Layers and Dropout layers with the ratio of 20% in order to achieve the objective of classification.

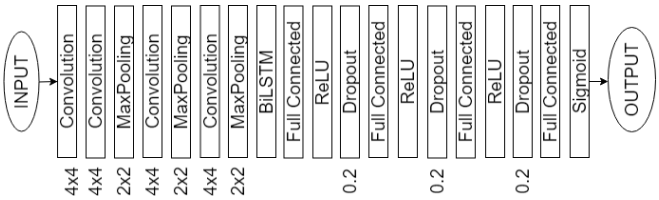

Fig. 5. CNN Model

C. HTML Builder

Recognized components were successfully translated into HTML code via the bootstrap framework. It was performed with the help of the coordinates from the result of the contour finding algorithms. Fig. 6 demonstrates the latest output that obtained from a browser when the input image is the first one of the images in Fig. 1.

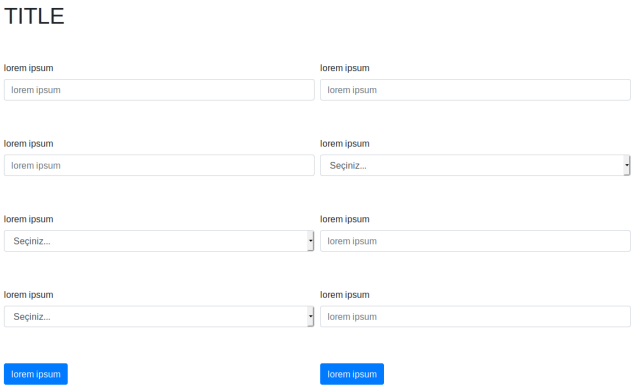

Fig. 6. Browser render of the design code produced by HTML Builder algorithm from sketch input image

As seen in the HTML builder algorithm shown in Fig. 7, first we created the templates for a header and a footer. Second, we detected how many items there are on each of the rows with coordinates of the components. Then we mapped the labels of the components to their template codes. At the end of this process the body section of the HTML code was successfully obtained. Finally the header, body and footer sections were combined as well. So, the final HTML code was composed.

The algorithm that developed for the purpose of performing HTML transformation has been demonstrated in Fig. 7

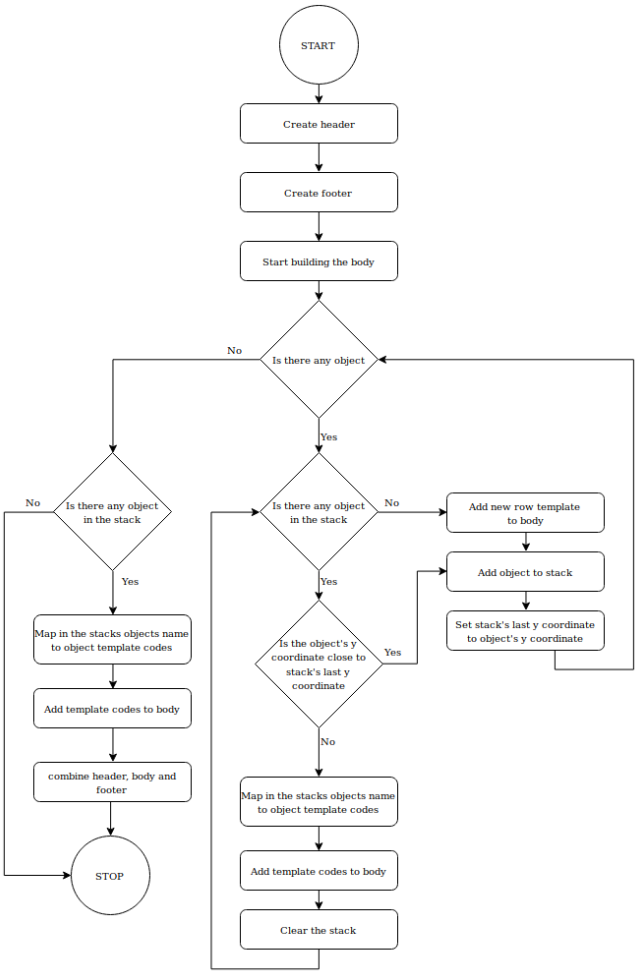

Fig. 7. HTML builder algorithm

V. Results

In this study, the purpose of generating automatic HTML code from mock-up images was successfully reached. For this purpose, four consecutive principal stages were carried out such as object detection, object cropping, object recognition, and HTML building. These successive steps are the milestones of our proposed algorithm. A CNN architecture was also used to be utilized in the object recognition stage of the study. Thus, various morphological transformations and deep learning which is highly popularized in recent years were performed. At last, another algorithm for HTML code generation, which is the final stage of the proposed algorithm, was presented. As a result of 200 epoch training of the model shown in Fig. 5 using the one hot encoding method accuracy and validation accuracy were obtained as 0.96 and 0.73 respectively. Fig. 6 shows the browser-interpreted display of the design code produced by the HTML Builder algorithm.

VI. Conclusions

Converting web page mock-ups to their mark-up code with minimum time and labor cost has become a significant topic in recent years when the artificial intelligence has been rapidly revolutionizing the industry by entering almost every field.

In this study, we have developed a system that takes hand-drawn web page mock-ups and gives a structured HTML code. To that end, a dataset consisting of images containing various hand-drawn sketches of web page designs was used. This dataset, which consists of 186 samples in total, has also been utilized in the creation of a corresponding dataset that contains the components contained in each image. Thus, the dataset, which was created by grouping all the components in 4 different classes, was used as training data for the CNN model to perform the process of object recognition.

In this study, the components in the picture were cropped by performing object detection with image processing techniques. It was determined which components were obtained by our trained CNN model. Finally, the purpose of generating HTML code was achieved using our HTML builder script with the help of the coordinates came from the algorithms of contour finding.

As a result, after the training phase of 200 epoch, accuracy and validation accuracy were obtained as 96% and 73%, respectively.

References

[1] Sketch2code. Microsoft AI Labs. [Online]. Available: https://github.com/Microsoft/ailab/tree/master/Sketch2Code/model/images

[2] T. A. Nguyen and C. Csallner, "Reverse Engineering Mobile Application User Interfaces with REMAUI (T)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, nov 2015, pp. 248–259. [Online]. Available: http://ieeexplore.ieee.org/document/7372013/

[3] S. Natarajan and C. Csallner, "P2A: A Tool for Converting Pixels to Animated Mobile Application User Interfaces," *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems - MOBILESoft '18*, pp. 224–235, 2018. [Online]. Available: http://dl.acm.org/citation.cfm?doid=3197231.3197249

[4] T. Beltramelli, "pix2code: Generating code from a graphical user interface screenshot," *CoRR*, vol. abs/1705.07962, 2017. [Online]. Available: http://arxiv.org/abs/1705.07962

[5] K. P. Moran, C. Bernal-Cárdenas, M. Curcio, R. Bonett, and D. Poshyvanyk, "Machine learning-based prototyping of graphical user interfaces for mobile apps," *IEEE Transactions on Software Engineering*, pp. 1–1, 2018.

[6] [Online]. Available: https://github.com/

[7] S. P. Reiss, Y. Miao, and Q. Xin, "Seeking the user interface," *Automated Software Engineering*, vol. 25, no. 1, pp. 157–193, mar 2018. [Online]. Available: https://doi.org/10.1007/s10515-017-0216-3