Style-Aware Sketch-to-Code Conversion for the Web

TOMMASO CALÒ, Politecnico di Torino, Italy
LUIGI DE RUSSIS, Politecnico di Torino, Italy

While sketching a graphical user interface (GUI) is a necessary step towards the creation of a Web application, its transformation into a coded GUI, with the proper styles, is still a tedious and time-consuming task that a designer should perform. Recently, a set of Machine Learning techniques has been applied to automatically generate code from sketches to ease this part of the design process. These techniques effectively convert the sketches into a skeleton structure of the GUI but are not designed to consider the styles to be applied to the generated HTML page. Moreover, having the possibility to explore different styles, starting from a sketch, might further support the designer in their work. In this paper, we move the first steps to enable this opportunity by proposing a method that allows the designer to input the sketch of the Web-based GUI and select a reference style to be applied. Our method automatically injects the reference styles into the sketch components and then uses an automatic code generation technique to obtain the final code. Preliminary experiments carried out with the navigation bar component show the effectiveness of the proposed method.

CCS Concepts: • **Human-centered computing → Graphical user interfaces**; *Interface design prototyping*; • **Computing methodologies → Machine learning**; **Computer vision**.

Additional Key Words and Phrases: machine learning, web elements, user interface, convolutional neural network

ACM Reference Format:
Tommaso Calò and Luigi De Russis. 2022. Style-Aware Sketch-to-Code Conversion for the Web. In *EICS 2022: ACM SIGCHI Symposium on Engineering Interactive Computing Systems, June 21–24, 2022, Sophia Antipolis, France.* ACM, New York, NY, USA, 6 pages. https://doi.org/XXXXXXX.XXXXXXX

1 INTRODUCTION

When designers first start thinking about a graphical user interface (GUI), they often sketch rough pictures of the screen layouts. Their initial goal is to work on the overall layout and structure of the components, rather than to refine the detailed look-and-feel. Designers use these sketches and other low-fidelity techniques to quickly consider design ideas, later shifting to interface construction tools or handing off the design to a programmer. However, transitioning from those sketches to a coded interface with a suitable look-and-feel is still a manual and time-consuming task [11, 12].

Supporting this transition is challenging due to the diversity of sketches and the complexity of coded GUIs. Therefore, it is of high interest for the research community to find methods and tools able to support designers in the process of translating between prototypes of the user interface. Several research projects, indeed, have tried to automate this translation. For instance, Beltramelli [1] proposed Pix2code, an end-to-end approach based on Convolutional and Recurrent Neural Networks that allows the generation of code from a mock-up screenshot taken as an input. Robinson [8], instead, presented sketch2code, a system to automatically transform hand-drawn sketches into coded GUIs. Both these works capture well the backbone structure of the GUI and translate it into code, but they are not designed to consider the *aesthetics* (e.g., colors and shadows) to be applied to the generated interface, which remains a manual and expensive task. Indeed, having the chance to explore different styles, immediately after creating a sketch, might further empower the designer in their creative work.

In this paper, we put forward a novel approach where we focus on translating a sketch of a Web interface to the related code, letting the designer also choose the *style* of the generated elements from an arbitrarily chosen picture, such as artwork or an infographic. To do so, we segment the input sketch to derive the single components of the web-based interface and their positions. Then, for the derived components, the designer can choose a reference style image which is used to guide the choice of the style of the input sketch. Given its complexity, the problem of style selection is split into two sub-problems: a) the selection of colors, accomplished with a clustering-based technique that extracts the most prominent colors in the reference image, and b) a feature distance-based metric technique for selecting the style of the text. For the purpose of this paper, we experiment with this approach with a single component, the *navigation bar*, widely present on many websites. Findings show that our method is able to select effectively the style of the reference template for the sketched component, showing good results in the color and in the text style selection, which well resembles the referenced style image.

2 BACKGROUND AND RELATED WORKS

Although the generation of computer programs is an active research field, program generation from visual inputs (like sketches) is still a relatively underexplored area. The problem of generating code from visual inputs is strictly related to the problem of automatically reverse-engineering GUIs: reverse-engineering approaches are mainly applied to generate code from GUI mock-ups or screenshots. Nguyen and Csallner [7], for example, developed a method to reverse-engineering Android user interfaces from screenshots. However, their method heavily relies on heuristics and expert knowledge to be implemented successfully, so its applicability is restricted to a limited domain of interfaces.

Similar approaches have been used to create tools able to generate code from hand-drawn wireframes. These tools [5, 13] are useful to designers who wish to quickly sketch and prototype possible UI layouts. A more complex version of this task is generating code from complete UI screenshots, as it requires that the system handle the stylistic and structural variation present in real-world app screens. Pix2code [1] was one of the first works attempting to address the problem of GUI code generation from visual inputs by leveraging machine learning to learn latent variables instead of engineering complex heuristics. To exploit the graphical nature of the input, Pix2code approaches the problem of converting screenshots to code as an image captioning problem; the author implemented first a Convolutional Neural Network (CNN) [4] performing unsupervised feature learning, mapping the raw input image to a learned representation, and then a Recurrent Neural Network (RNN) [9] performing language modeling on the textual description associated with the input picture. In this latter step, at every iteration, the screenshot of the interface is concatenated to the latent state variable of the RNN, and the final output is a set token that can be parsed into the final code of the interface. The Pix2code model shows good generalization abilities even with out-of-domain samples. UI2Code [2] uses a similar architecture to generate a GUI skeleton from a screenshot that describes the relative positioning of GUI elements.

Another work very close to ours is Sketch2Code [8]. Sketch2Code approaches the problem similarly to Pix2code, with the difference that the author trains the model on a specially-prepared dataset of GUI sketch images. In addition, the author does not implement a language model and, instead, uses only a CNN to identify the application components. Sketch2Code, then, represents every component as a JavaScript Object Notation (JSON) structure, which is then parsed by a GUI parser to create platform-specific code.

As depicted, none of these previous works focus their attention on the automatic style customization of graphical elements. Our approach aims at filling this gap by introducing an approach that is similar to Sketch2Code in the fact that we start from sketches, but differs from Pix2Code since we do not implement any language model to perform the translation. Instead, we implement a CNN to infer the structural properties of the sketched Web component. We then use an automatic procedure to select the stylistic properties from a freely chosen image and generate the final code with the help of a parser.

3 METHODOLOGY

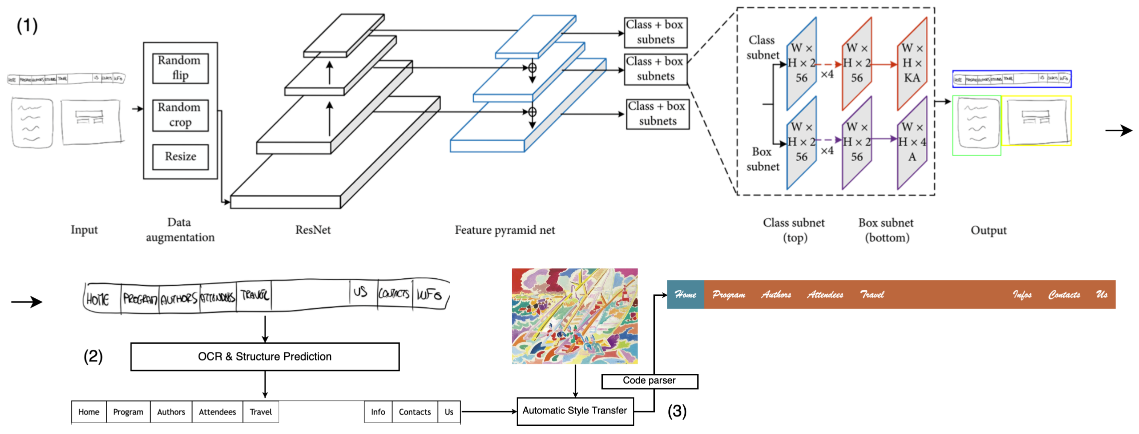

Fig. 1. Method overview. (1) Starting from the sketch of a webpage, we perform segmentation of its interface. (2) We infer the structure and the textual elements of the selected component (Section 3.1). (3) Style properties of the reference image are extracted and injected into the structural properties of the sketch. Finally, a parser generates the final code along with the rendering of the component (Sections 3.2 and 3.3).

The task of generating the computer code of a styled interface from a sketch can be split into three sub-problems. First, the problem of understanding the sketch and inferring which are the present elements and their positions. Second, for every component recognized on the sketch, apply a specific style, given a reference style. Finally, the last challenge is to generate the code of the resulting styled component. The presented approach has been implemented using Python 3.8. The neural network has been implemented using PyTorch, the images processing with PIL and OpenCV, while the data processing has been conducted with Pandas and Numpy.

3.1 Sketch Understanding

The task of understanding the sketch is a computer vision task that, given the sketch of a Web-based GUI, consists of the detection and identification of the included components (e.g., buttons, navbars, etc.) and their relative position. For this task, we adopted the same method of Sketch2Code [8], which uses RetinaNet [6], a popular single-stage detector that is accurate and runs fast. RetinaNet uses a feature pyramid network to efficiently detect objects at multiple scales and introduces a new loss, the focal loss function, to alleviate the problem of the extreme foreground-background class imbalance. RetinaNet can simultaneously predict both the class and the box position of the object under detection. For performing this task, we use the same dataset presented in Sketch2Code. To recognize written words in the sketch, in addition, we utilize an OCR technique.

3.2 Automatic Style Transfer

Given that a Web component is a complex aggregation of multiple visual elements, we split the problem of extracting the style from the selected reference image into two sub-problems, *color extraction* and *text style selection.*

Color Extraction. We extract the most dominant colors from the selected reference image by using a median cut based clustering technique, which works by sorting data of an arbitrary number of dimensions into series of sets by recursively cutting each set of data at the median point along the longest dimension, in our case the color dimensions in the image (i.e., the RGB channels present in a colored picture).

Text Style Selection. To select that text style that mostly resembles the selected reference image we exploit the feature extraction power of CNNs. In detail, given a list of preferred fonts, we pass both the image and a sample sentence for each font through a pre-trained Visual Geometry Group (VGG) neural network. We then choose the font which minimizes the cosine similarity between the two hidden representations. Mathematically, given the model S, a reference image $i \in I$ and a set of fonts F, we choose the j-th style $f \in F$ such that

$$j = \arg\min_{j} S(i) * S(f_j) \tag{1}$$

where $S(i)$ and $S(f_j)$ are the last layer activation values of the network. The formula above ensures that the selected reference image and the font have some visual similarity in common, or at least that the similarity is maximal among the chosen fonts.

3.3 Code Generation and UI Reconstruction

Given the customized component obtained by the previous step, we pass it again through a multi-headed VGG Convolutional Neural Network [10] to infer its characteristics, both in structure and style. The resulting object is then passed to an external parser, along with the relative box positions obtained from the sketch understanding phase (Section 3.1) to produce the final code representation of the GUI. The external parser maps the feature inferred from the CNN to both CSS attributes for style features and structural HTML attributes for content features. Finally, an algorithm embeds them into a website template to generate the final interface.

4 PRELIMINARY RESULTS

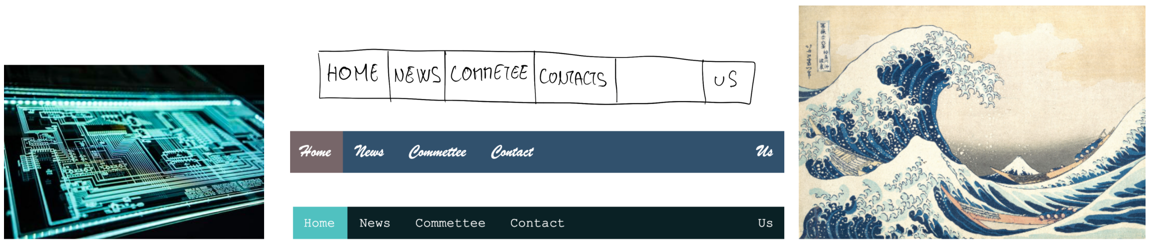

Fig. 2. Results of the automatic style transfer algorithm and the UI reconstruction. The sketch above is converted into the corresponding navbars having the styles of the images on the sides.

To validate our method, our experiments focus mainly on testing the correct prediction of the structure of the sketch with both synthetic and real sketches, having correct visual feedback from the extraction of dominant colors from the reference image, and validating the robustness of the choice of the text font that most resembles the style of the reference image through various sentence samples. Since the segmentation and reconstruction methods were adopted from an already validated by Sketch2Code [8] and UICode [2] we do not report the performance of those methods in this paper. The experiments have been conducted with Google Colab.

Dataset. To test the effectiveness of the style transfer technique we built a synthetic dataset of 3,000 navigation bars (navbars) sketches, the navbars can have at most five items floating left and three items floating right; the aim of the structure prediction model is to infer the number of right and left items. In addition, we fine-tuned the resulting model to 50 navbars sketches to evaluate the performances of the model in a real scenario.

Measures. The performance of the method we propose relies on two main objectives.

(1) The convolutional neural network must be able to classify correctly the structural features of the sketched component in order to parse them into code.
(2) The color extraction algorithm must give correct visual feedback on the capabilities of extracting the right colors and choosing the right font given the reference image.

For the former, we evaluate the classification performances of the CNN on the synthetic and on the real sketches dataset. So that, given a sketch, the network must properly infer its content features. To evaluate its performance, we use the accuracy of the prediction with respect to the ground truth, which represents the items positions in the sketched component. The accuracy is calculated as the number of sketches for which the model correctly predicts *all* the structural components of the sketch. For the second objective, we visually evaluate the performances of the color extraction algorithm in the results we obtained, leaving for future works a more extended evaluation with designers.

Experiments. To evaluate the performance of the CNN for the *sketch structure prediction*, we split the synthetic sketch dataset into 2,500 train samples and 500 test samples, we trained the network 20, 30, and 50 epochs with pre-trained weights on ImageNet [3] and we then tested fine-tuned the network on 50 real sketches, and tested on 20.

Epochs	Synthetic Sketches	Real Sketches
20	0.898	0.678
30	0.909	0.685
50	0.912	0.691

Table 1. Accuracy in the predictions of the structural features of the synthetic and real sketches.

As reported in Table 1, the performances of the convolutional network in distinguishing the structural features of the sketched component achieve very good results with a top 0.691 accuracy over the real sketches set after 50 epochs training. Finally, the *color extraction* algorithm has been evaluated on 10 reference images, while the *Text Style Selection* algorithm has been evaluated on 5 different fonts. The resulting visualization shows encouraging performance for the proposed method to capture the style as well as the color of the reference image. In further work, we plan to conduct an extensive user study to evaluate our results.

5 CONCLUSIONS AND FUTURE WORKS

This paper presents a method to support designers in generating web pages from a sketch with the addition of style. Our approach consists of three main parts: a deep learning architecture for segmentation and classification, a style extraction procedure from a reference image, and a parsing algorithm. Among its advantages, it is fully integrated and easily adaptable for different sketches of different domains. Then, to our knowledge, it is the first model that allows the designer to personalize the style of the sketched component automatically from a selected template. Lastly, it is highly modular, as changing a single module does not require changing the precedent components.

Our approach has some limitations that could eventually be addressed in future research. First of all, both the style and content features of the components are handcrafted thus the model cannot generalize out of the sketching specification. This is done to obtain good results due to the complexity of style specifications in web components. Future work is to implement techniques that allow visual style transfer with language models instead of procedural methods since language models can generalize out of handcrafted features in this specific task, e.g., as shown by Beltramelli [1]. Secondly, the automatic style transfer technique is limited to predefined stylistic properties, in our case, colors and fonts. In a real web design scenario, there are many more stylistic properties to take into account, such as shadows, borders, and dynamics of responsive elements. With such characteristics, it could become challenging to apply our method extensively in real-world applications. Future research should focus on automatic models that could handle such complexity in an integrated fashion. Finally, the proposed method needs to be tested with a diverse set of sketches and web elements, as well as to be included in a tool for designers, where they can select different styles to be applied to their own sketches. Such a tool will, then, be evaluated in user studies to assess the usefulness of the overall approach.

To conclude, sketch-to-code translation of user interfaces is closer to being implemented in real-world applications, and our work is a first attempt to allow an automatic stylization of GUI elements leveraging machine learning techniques, to deliver a more integrated approach, able to support designers in easing this time-consuming part of their work.

REFERENCES

[1] Tony Beltramelli. 2018. Pix2code: Generating Code from a Graphical User Interface Screenshot. In *Proceedings of the ACM SIGCHI Symposium on Engineering Interactive Computing Systems* (Paris, France) *(EICS '18)*. Association for Computing Machinery, New York, NY, USA, Article 3, 6 pages. https://doi.org/10.1145/3220134.3220135

[2] Chunyang Chen, Ting Su, Guozhu Meng, Zhenchang Xing, and Yang Liu. 2018. From UI Design Image to GUI Skeleton: A Neural Machine Translator to Bootstrap Mobile GUI Implementation. In *The 40th International Conference on Software Engineering, Gothenburg, Sweden*. ACM.

[3] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. 2009. Imagenet: A large-scale hierarchical image database. In *2009 IEEE conference on computer vision and pattern recognition*. Ieee, 248–255.

[4] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. 2014. Large-scale Video Classification with Convolutional Neural Networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

[5] James A. Landay. 1996. SILK: Sketching Interfaces like Krazy. In *Conference Companion on Human Factors in Computing Systems* (Vancouver, British Columbia, Canada) *(CHI '96)*. Association for Computing Machinery, New York, NY, USA, 398–399. https://doi.org/10.1145/257089.257396

[6] Tsung-Yi Lin, Priya Goyal, Ross B. Girshick, Kaiming He, and Piotr Dollar. 2020. Focal Loss for Dense Object Detection. *IEEE Transactions on Pattern Analysis & Machine Intelligence* 42, 02 (feb 2020), 318–327. https://doi.org/10.1109/TPAMI.2018.2858826

[7] Tuan Anh Nguyen and Christoph Csallner. 2015. Reverse Engineering Mobile Application User Interfaces with REMAUI. In *Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering* (Lincoln, Nebraska) *(ASE '15)*. IEEE Press, 248–259. https://doi.org/10.1109/ASE.2015.32

[8] Alex Robinson. 2019. Sketch2code: Generating a website from a paper mockup. arXiv:1905.13750 [cs.CV]

[9] David E. Rumelhart and James L. McClelland. 1987. *Learning Internal Representations by Error Propagation*. 318–362.

[10] Karen Simonyan and Andrew Zisserman. 2015. Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv:1409.1556 [cs.CV]

[11] Sarah Suleri, Vinoth Pandian Sermuga Pandian, Svetlana Shishkovets, and Matthias Jarke. 2019. Eve: A Sketch-Based Software Prototyping Workbench. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems* (Glasgow, Scotland Uk) *(CHI EA '19)*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3290607.3312994

[12] Miriam Walker, Leila Takayama, and James A. Landay. 2002. High-Fidelity or Low-Fidelity, Paper or Computer? Choosing Attributes when Testing Web Prototypes. *Proceedings of the Human Factors and Ergonomics Society Annual Meeting* 46, 5 (2002), 661–665. https://doi.org/10.1177/154193120204600513

[13] Benjamin Wilkins. 2017. Airbnb Sketching Interfaces. https://airbnb.design/sketching-interfaces. Accessed: 2022-01-10.